Machine Learning: Models And Algorithms

Daniel Bloch

1st of August 2021

The copyright to this computer software and documentation is the property of Quant Finance Ltd. It may be used and/or copied only with the written consent of the company or in accordance with the terms and conditions stipulated in the agreement/contract under which the material has been supplied.

Copyright © 2021 Quant Finance Ltd
Quantitative Analytics, London

Created: 1 August 2021

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Machine Learning: Models And Algorithms

Daniel BLOCH [1]
QUANT FINANCE
Working Paper

7th of January 2019
Revised on the 1st of August 2021
Revised version: 1.2.5

[1] dan@quantfin.eu

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Abstract

This textbook introduces the mathematical models and algorithms utilised in machine learning, covering supervised and unsupervised learning as well as reinforcement learning: In supervised learning we present ensemble models, artificial neural networks, deep neural networks, recurrent neural networks and associative reservoir computing, while in unsupervised learning we present radial basis function (RBF), recurrent RBF networks amongst others. In both cases we give examples for chaotic series prediction as well as financial time series. We then introduce the learning theory and formalise supervised learning in the cases of neural networks and recurrent neural networks. We present statistical inference and graph theory in order to introduce probabilistic networks, such as generative models, random graphs, graphical models and complex networks. We discuss the modelling of sequential data and its application to sequence to sequence learning in natural language (NL). We introduce the Transformer model and discuss pre-training techniques in language understanding and language generation. Using stochastic control theory and dynamic programming, we introduce temporal difference (TD) learning and its extensions, discuss Deep BSDE, present reinforcement learning (RL) and deep RL, detailing various algorithms for learning the optimal policy. Then, we introduce constrained optimisation and global search optimisation, and discuss the use of machine learning for solving some continuous and discrete optimisation problems. Finally, we apply supervised and reinforcement learning to the pricing and hedging of option prices.

Machine Learning, Supervised Learning, Unsupervised Learning, Reinforcement Learning, Ensemble Models, Artificial Neural Networks, Deep Neural Networks, Recurrent Neural Networks, Associative Reservoir Computing, Probabilistic Networks, Complex Networks, Multifractal Networks, Natural Language Processing, TD-Learning, Deep BSDE, Constrained Optimisation, Global Search Optimisation, Stochastic Control, Dynamic Programming, Option Pricing and Hedging

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

I would like to thank my wife and children for their patience and support during this adventure.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Contents

I Knowledge representation **36**

1 Presentation **37**
1.1 Models . . . 37
1.1.1 Mathematical models . . . 37
1.1.1.1 Description . . . 37
1.1.1.2 Statistical model . . . 39
1.1.1.3 Defining a theory . . . 39
1.1.1.4 Mathematical representation . . . 40
1.1.2 Algorithms . . . 41
1.1.2.1 An overview . . . 41
1.1.2.2 Description . . . 42
1.1.2.3 Classification . . . 43
1.1.3 Computing . . . 44
1.1.3.1 Description . . . 44
1.1.3.2 Control system . . . 45
1.1.3.3 Computer cluster . . . 46
1.1.3.4 Orchestration . . . 46
1.1.3.5 Research . . . 48
1.1.3.6 Knowledge representation . . . 48
1.2 Machine learning . . . 49
1.2.1 An overview . . . 49
1.2.1.1 Artificial intelligence . . . 49
1.2.1.2 Towards machine learning . . . 49
1.2.2 Objectives . . . 51
1.2.2.1 Life is a sub-optimal optimisation problem . . . 51
1.2.2.2 Defining intelligence . . . 51
1.2.2.2.1 Human intelligence . . . 51
1.2.2.2.2 Noisy behaviour . . . 52
1.2.2.2.3 Machine intelligence . . . 52
1.2.2.3 Accomplishing complex goals . . . 52
1.2.2.3.1 Solving tasks . . . 52
1.2.2.3.2 Replicating human intelligence . . . 53
1.2.2.4 Accounting for nonlinear information . . . 53
1.2.3 Description . . . 54
1.2.3.1 Knowledge representation . . . 54
1.2.3.2 The learning . . . 55
1.2.3.2.1 Online and offline learning . . . 55
1.2.3.2.2 The types of learning . . . 55

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.2.3.2.3 Some tasks . . . 56
1.2.3.3 Attention mechanisms . . . 56
1.2.3.4 Control architectures . . . 57
1.2.3.4.1 Some terminologies . . . 57
1.2.3.4.2 Finite-state machine . . . 58
1.2.3.4.3 Behaviour tree . . . 59
1.2.3.5 Planning . . . 61
1.2.3.5.1 Description . . . 61
1.2.3.5.2 Domain modelling . . . 63
1.2.3.5.3 Some algorithms . . . 63
1.2.4 Some properties . . . 64
1.2.4.1 Prediction . . . 64
1.2.4.2 Optimisation . . . 64
1.2.4.3 Generalisation . . . 64
1.2.4.4 Limitation . . . 65
1.2.5 Processes and techniques . . . 65
1.2.6 Some models . . . 67
1.2.7 Problems solving . . . 68
1.2.8 Applications to the industry . . . 69
1.2.8.1 Defining AI strategy and production readiness . . . 70
1.2.8.2 Some examples . . . 72
1.2.8.2.1 Energy . . . 72
1.2.8.2.2 Event management . . . 73
1.2.8.2.3 Pharmacogenomics . . . 73
1.2.8.2.4 Finance . . . 74
1.3 Neural networks . . . 75
1.3.1 An overview . . . 75
1.3.1.1 A brief history . . . 75
1.3.1.2 A Description . . . 75
1.3.2 Modelling changing environment . . . 76
1.3.2.1 Multiple characteristic timescales . . . 76
1.3.2.2 Long range dependence . . . 77
1.3.2.3 The variable weight neural network . . . 77
1.3.3 Interpreting neural networks . . . 78
1.3.3.1 Explaining the input feature importance . . . 78
1.3.3.2 Determining the contribution of explanatory variables . . . 79
1.3.4 Optimisation for machine learning . . . 79
1.3.4.1 Function estimation . . . 79
1.3.4.2 Parameters estimation . . . 80
1.4 Computer vision . . . 81
1.4.1 Description . . . 81
1.4.1.1 An overview . . . 81
1.4.1.2 Some tasks . . . 82
1.4.1.3 Computer vision systems . . . 83
1.4.2 Digital images . . . 84
1.4.2.1 Description . . . 84
1.4.2.2 The categories . . . 85
1.4.2.2.1 Bitmap . . . 85
1.4.2.2.2 Vector graphics . . . 85
1.4.2.3 Format . . . 85

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.4.3 Visual detection . . . 86
1.4.3.1 Overview . . . 86
1.4.3.2 Description . . . 86
1.4.3.3 Some detection system . . . 87
1.4.3.3.1 YOLO detection system . . . 87
1.4.3.3.2 RetinaNet detection system . . . 88
1.4.4 Parsing documents . . . 88
1.4.4.1 Terminology . . . 88
1.4.4.1.1 XML . . . 88
1.4.4.1.2 JSON . . . 88
1.4.4.1.3 Parsing . . . 89
1.4.4.2 OCR . . . 89
1.4.4.3 PDF . . . 90
1.4.4.3.1 Content . . . 90
1.4.4.3.2 Imaging model . . . 91
1.4.4.3.3 Text . . . 91
1.4.4.3.4 Fonts . . . 91
1.4.4.3.5 Conversion . . . 91
1.4.4.4 PDF parser . . . 91
1.4.4.4.1 Description . . . 91
1.4.4.4.2 Fully automated PDF data extraction software . . . 92
1.4.4.4.3 Repetitive information . . . 92
1.4.4.4.4 Abbyy FineReader . . . 92
1.4.4.5 Extracting tables . . . 92
1.4.4.5.1 Towards a solution . . . 93
1.4.4.5.2 The difficulty of getting a solution . . . 93
1.5 Natural language processing . . . 93
1.5.1 An overview . . . 94
1.5.1.1 Natural languages . . . 94
1.5.1.2 Representational systems for NLP . . . 95
1.5.1.3 A brief history . . . 95
1.5.2 NLP tasks . . . 96
1.5.2.1 Syntax . . . 96
1.5.2.2 Semantics . . . 98
1.5.2.3 Discourse . . . 101
1.5.2.4 Speech . . . 101
1.5.3 Some example tasks . . . 102
1.5.3.1 OCR . . . 102
1.5.3.2 Named-entity recognition . . . 104
1.5.3.3 Paraphrasing and textual entailment . . . 105
1.5.3.4 Natural language understanding . . . 105
1.5.4 Contextual word representations . . . 106
1.5.4.1 Word vectors . . . 106
1.5.4.2 Contextual word vectors . . . 108
1.5.5 Transfer learning . . . 110
1.5.5.1 Definition . . . 110
1.5.5.2 Pretraining . . . 111
1.5.5.3 Adaptation . . . 113
1.5.5.3.1 Architectural modifications . . . 113
1.5.5.3.2 Optimisation schemes . . . 113

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.5.5.3.3 Practical considerations . . . 114
1.5.5.3.4 Getting more signal . . . 114
1.5.6 Sentence similarity search . . . 114
1.5.6.1 Description . . . 114
1.5.6.2 Some models . . . 115
1.5.6.3 Examples . . . 117
1.5.6.4 Extending the model . . . 118
1.5.6.5 Transfer learning . . . 118
1.6 The learning process . . . 119
1.6.1 Overview . . . 119
1.6.1.1 Classification and prediction tasks . . . 119
1.6.1.2 Datasets . . . 120
1.6.1.2.1 Computer vision . . . 120
1.6.1.2.2 NLP . . . 120
1.6.2 Training and exploitation of the model . . . 121
1.6.2.1 Sample hashing . . . 121
1.6.2.2 Cross-validation . . . 122
1.6.2.3 Variations to cross-validation . . . 123
1.6.2.4 Bootstrapping . . . 125
1.6.3 Financial example . . . 125
1.6.3.1 The framework . . . 125
1.6.3.1.1 The input variables . . . 125
1.6.3.1.2 The data . . . 125
1.6.3.1.3 Evaluation methods . . . 126
1.6.3.1.4 The measures . . . 126
1.6.3.2 Using neural networks to model returns . . . 126
1.6.3.2.1 An overview . . . 126
1.6.3.2.2 Combining neural networks with evolutionary algorithms . . . 127
1.6.3.3 The learning process . . . 127
1.7 ML pipeline . . . 128
1.7.1 Overview . . . 128
1.7.2 Some platforms . . . 129
1.7.2.1 Build all custom . . . 130
1.7.2.2 ML models libraries . . . 130
1.7.2.3 ML models software . . . 131
1.7.2.4 End-to-end solutions . . . 133
1.7.3 Production framework . . . 135
1.7.3.1 Terminology . . . 135
1.7.3.1.1 Cluster manager . . . 135
1.7.3.1.2 Big data tools . . . 137
1.7.3.1.3 Pipeline creator: workflow manager . . . 139
1.7.3.1.4 Serverless platform . . . 140
1.7.3.1.5 Notebook . . . 140
1.7.3.2 The platform . . . 141
1.7.3.2.1 Data pipeline management . . . 141
1.7.3.2.2 Data infrastructure . . . 142
1.7.3.2.3 ML model infrastructure . . . 142

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2 Information and knowledge **144**
2.1 Algebraic logic . . . 144
2.1.1 Boolean algebra . . . 144
2.1.1.1 An overview . . . 144
2.1.1.2 Description . . . 145
2.1.1.2.1 Some operations . . . 145
2.1.1.2.2 Some laws . . . 147
2.1.1.2.3 Representations . . . 148
2.1.1.2.4 About algebras . . . 148
2.1.1.2.5 Propositional logic . . . 149
2.1.2 Fuzzy logic . . . 150
2.1.2.1 An overview . . . 150
2.1.2.2 Description . . . 150
2.1.2.2.1 The process . . . 150
2.1.2.2.2 The operators . . . 151
2.1.2.2.3 Fuzzy sets . . . 152
2.1.2.2.4 Defuzzification . . . 152
2.1.2.2.5 Difference with probability . . . 153
2.2 Introduction to information theory . . . 153
2.2.1 Presenting a few concepts . . . 153
2.2.1.1 Uncertainty . . . 153
2.2.1.2 Coding . . . 154
2.2.1.3 Randomness . . . 154
2.2.2 Some facts on entropy in information theory . . . 155
2.2.3 Relative entropy and mutual information . . . 157
2.2.4 Bounding performance measures . . . 158
2.2.5 Feature selection . . . 160

3 Data **164**
3.1 Introduction to data . . . 164
3.1.1 From data to information . . . 164
3.1.1.1 Data . . . 164
3.1.1.1.1 Definitions . . . 164
3.1.1.1.2 Dataset . . . 164
3.1.1.2 Features . . . 165
3.1.1.2.1 Definition . . . 165
3.1.1.2.2 Feature engineering . . . 165
3.1.2 Data representation . . . 166
3.1.2.1 Units of information . . . 166
3.1.2.2 Data structures . . . 167
3.1.2.2.1 Bit array . . . 168
3.1.3 Character encoding . . . 168
3.1.3.1 Character representation . . . 168
3.1.3.2 Unicode encoding . . . 169
3.1.3.3 One-hot encoding . . . 170
3.2 Data storage . . . 172
3.2.1 Definitions . . . 172
3.2.1.1 Database model . . . 172
3.2.1.2 Database . . . 172
3.2.2 Relational database . . . 175

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.2.2.1 Description . . . 175
3.2.2.2 Database normalisation . . . 177
3.2.2.3 Storing tables . . . 178
3.2.2.4 Column-oriented DBMS . . . 178
3.2.3 Graph database . . . 179
3.2.3.1 Description . . . 179
3.2.3.2 Relational versus graph database . . . 181
3.2.4 Big data . . . 182
3.2.4.1 Definitions . . . 182
3.2.4.2 Data integration . . . 184
3.2.4.2.1 Description . . . 184
3.2.4.2.2 Smarter clouds . . . 185
3.3 Introduction to data mining . . . 187
3.3.1 Data mining . . . 187
3.3.2 Data preparation . . . 189
3.3.2.1 Data collection . . . 189
3.3.2.2 Data pre-processing . . . 190
3.3.3 Knowledge discovery . . . 191
3.3.3.1 Pattern recognition . . . 191
3.3.3.2 Predictive modelling . . . 192
3.3.3.3 Model training . . . 193
3.3.4 Clustering . . . 193
3.3.4.1 Introduction . . . 193
3.3.4.1.1 Overview . . . 193
3.3.4.1.2 Some algorithms . . . 194
3.3.4.2 Probabilistic models . . . 194
3.3.4.3 Features representation . . . 195
3.3.5 Classification . . . 196
3.3.5.1 Definitions . . . 196
3.3.5.2 Some examples . . . 197
3.3.5.2.1 Anomaly detection . . . 197
3.3.5.3 The algorithms . . . 198
3.3.6 The challenges of computational learning . . . 199
3.3.7 Evaluation and validation . . . 201
3.3.7.1 Overview . . . 201
3.3.7.2 Measures of performance . . . 203
3.3.8 Representation learning . . . 207
3.4 Data management . . . 207
3.4.1 Overview . . . 207
3.4.1.1 A brief review . . . 207
3.4.1.2 Some challenges . . . 209
3.4.1.2.1 Missing data . . . 209
3.4.1.2.2 Duplicate data . . . 209
3.4.1.2.3 Data variety . . . 210
3.4.1.2.4 Semantic data integration . . . 210
3.4.1.2.5 High-dimensional data space . . . 210
3.4.1.2.6 Data quality dimensions . . . 211
3.4.2 Data governance . . . 212
3.4.2.1 Data quality lifecycle . . . 212
3.4.2.2 Data management function . . . 216

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.4.2.3 A reference architecture . . . 216
3.4.3 Probabilistic and statistical models . . . 218
3.4.3.1 Data imputation . . . 218
3.4.3.2 Outliers . . . 219
3.4.3.3 Duplicate detection . . . 219
3.4.4 ML in data management . . . 220
3.4.4.1 Some examples . . . 220
3.4.5 Graph database . . . 221
3.5 Duplicates detection . . . 222
3.5.1 Introduction . . . 222
3.5.1.1 Description of the problem . . . 222
3.5.1.2 Overview . . . 225
3.5.1.3 Using graph theory . . . 226
3.5.2 Ad hoc techniques . . . 227
3.5.2.1 Character-based similarity metrics . . . 227
3.5.2.1.1 The edit distance . . . 227
3.5.2.1.2 The affine gap distance . . . 228
3.5.2.1.3 The Jaro distance metric . . . 228
3.5.2.1.4 Q-gram distance . . . 228
3.5.2.1.5 The Hamming distance . . . 228
3.5.2.2 Token-based similarity metrics . . . 228
3.5.2.2.1 Atomic strings . . . 229
3.5.2.2.2 WHIRL . . . 229
3.5.2.2.3 Q-Grams with tf.idf . . . 229
3.5.2.2.4 Hybrids . . . 229
3.5.3 Probabilistic matching models . . . 230
3.5.3.0.1 Bayes decision rule for minimum error . . . 230
3.5.3.0.2 Bayes decision rule for minimum cost . . . 231
3.5.3.0.3 Decision with a reject region . . . 232
3.5.4 Supervised and semisupervised learning . . . 232
3.5.4.1 Overview . . . 232
3.5.4.1.1 Supervised learning . . . 232
3.5.4.1.2 Semisupervised learning . . . 234
3.5.4.2 Examples . . . 234
3.5.4.2.1 Finding the optimal threshold . . . 234
3.5.4.2.2 Using NLP . . . 238
3.5.4.3 Graph-based approach . . . 240
3.5.5 Unsupervised learning . . . 241

4 Ensemble models **243**
4.1 The base models . . . 243
4.1.1 Introduction . . . 243
4.1.1.1 Presentation . . . 243
4.1.1.2 Definitions . . . 244
4.1.1.3 An optimisation problem . . . 245
4.1.1.4 Some characteristics . . . 246
4.1.1.4.1 Diversity . . . 246
4.1.1.4.2 Generalisation error . . . 246
4.1.1.4.3 Perturbing and combining . . . 247
4.1.2 Some classical models . . . 247

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.1.2.1 The k-nearest neighbour . . . 247
4.1.2.2 Binary space partitioning . . . 247
4.1.2.3 Decision tree . . . 249
4.1.2.3.1 Parse tree . . . 250
4.1.2.3.2 Regression . . . 250
4.1.2.3.3 Classification . . . 251
4.1.2.3.4 Selecting the splits . . . 251
4.1.2.4 Boosting . . . 252
4.1.2.4.1 The procedure . . . 252
4.1.2.4.2 Some algorithms . . . 253
4.1.2.4.3 Non-convex algorithms . . . 255
4.1.2.5 Bagging . . . 255
4.1.2.5.1 Description . . . 255
4.1.2.5.2 The procedure . . . 256
4.1.2.5.3 Algorithm for classification . . . 257
4.1.3 Bias and variance . . . 257
4.1.3.1 Regression . . . 257
4.1.3.2 Classification . . . 258
4.1.3.3 Instability . . . 259
4.1.4 Noise in random classification . . . 260
4.1.4.1 Convex potential functions . . . 260
4.1.4.1.1 Boosters . . . 260
4.1.4.2 A formalism . . . 261
4.1.4.2.1 Boostable . . . 261
4.1.4.2.2 Noise tolerance . . . 261
4.1.4.2.3 Random classification noise . . . 261
4.1.5 Random forest . . . 262
4.1.5.1 Tree bagging . . . 262
4.1.5.2 The procedure . . . 263
4.1.6 Gradient boosting . . . 264
4.1.6.1 Description . . . 264
4.1.6.2 Implementation . . . 265
4.1.6.3 Gradient tree boosting . . . 267
4.1.7 Towards combination methods . . . 267
4.1.7.1 Averaging . . . 268
4.1.7.1.1 Simple averaging . . . 268
4.1.7.1.2 Weighted averaging . . . 269
4.1.7.2 Voting . . . 270
4.1.7.2.1 Majority voting . . . 270
4.1.7.2.2 Plurality voting . . . 271
4.1.7.2.3 Weighted voting . . . 271
4.1.7.2.4 Soft voting . . . 272
4.2 Online learning and regret-minimising algorithms . . . 273
4.2.1 Simple online algorithms . . . 273
4.2.1.1 The Halving algorithm . . . 273
4.2.1.2 The weighted majority algorithm . . . 273
4.2.2 The online convex optimisation . . . 274
4.2.2.1 The online linear optimisation problem . . . 274
4.2.2.2 Considering Bergmen divergence . . . 275
4.2.2.3 More on the online convex optimisation problem . . . 276

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.3 Combining by learning . . . 277
4.3.1 Stacking . . . 277
4.3.2 Infinite ensemble . . . 279
4.3.3 Other methods . . . 280
4.3.3.1 The algebraic methods . . . 280
4.3.3.2 Mixture of experts . . . 280
4.3.4 Diversity measures . . . 281
4.3.4.1 Pairwise measures . . . 282
4.3.4.2 Non-pairwise measures . . . 283

II Neural Networks 285

5 Introduction to artificial neural networks 286
5.1 The classic networks . . . 287
5.1.1 The perceptron . . . 288
5.1.2 The Hopfield networks . . . 289
5.1.2.1 Description . . . 289
5.1.2.2 The energy function . . . 290
5.1.3 The multilayer perceptron . . . 291
5.1.3.1 Description . . . 291
5.1.3.2 A mathematical representation . . . 294
5.1.3.3 The universal approximation theorem . . . 296
5.1.4 Gradient descent and the delta rule . . . 297
5.1.4.1 The gradient descent rule . . . 297
5.1.4.2 The delta rule . . . 298
5.1.5 Accounting for time and earning profits . . . 299
5.1.5.1 Time factor . . . 299
5.1.5.2 Direction measures . . . 300
5.1.5.3 Time dependent direction profit . . . 301
5.2 Introducing multilayer networks . . . 302
5.2.1 Describing the problem . . . 302
5.2.2 Describing the algorithm . . . 303
5.2.3 Describing the nonlinear transformation . . . 304
5.2.4 A simple example . . . 305
5.3 Multi-layer back propagation . . . 306
5.3.1 The output layer . . . 307
5.3.2 The first hidden layer . . . 308
5.3.3 The next hidden layer . . . 310
5.3.4 Some remarks . . . 313
5.4 Summarising the feedforward ANN . . . 314
5.4.1 Forward pass . . . 314
5.4.2 Backward pass . . . 315
5.4.3 Summary . . . 316

6 Recurrent neural networks 317
6.1 Presenting recurrent neural networks . . . 317
6.1.1 An overview . . . 317
6.1.2 The algorithm . . . 318
6.1.2.1 Forward pass . . . 318

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

6.1.2.2 Backward pass . . . 319
6.1.3 The vanishing gradient problem . . . 320
6.1.4 Dealing with long-range dependencies . . . 321
6.1.4.1 Description . . . 321
6.1.4.2 An overview . . . 322
6.1.4.2.1 The LSTM . . . 322
6.1.4.2.2 Hessian Free Optimisation . . . 323
6.1.4.2.3 Reservoir computing . . . 323
6.1.4.2.4 Gradient norm clipping strategy . . . 323
6.1.5 Mathematical representation . . . 324
6.2 The long short-term memory . . . 326
6.2.1 The constant error carousel . . . 326
6.2.2 Network architecture . . . 327
6.2.2.1 Summary . . . 330
6.2.3 The learning algorithm . . . 330
6.2.3.1 Computational complexity of LSTM . . . 330
6.3 Reservoir computing . . . 332
6.3.1 Describing the Reservoir methods . . . 333
6.3.1.1 Description . . . 333
6.3.1.2 The parameters . . . 334
6.3.1.3 The readout . . . 334
6.3.1.4 Some comments . . . 336
6.3.2 Choosing the parameters . . . 336
6.3.2.1 Input Scaling . . . 336
6.3.2.2 Size of the reservoir . . . 337
6.3.2.3 Sparsity . . . 337
6.3.2.4 Spectral Radius . . . 337
6.3.2.5 Leak Rate . . . 338
6.3.3 Some improvements . . . 338
6.3.4 Empirical results . . . 339
6.4 Other networks . . . 346
6.4.1 Variable weights neural networks . . . 346
6.4.1.1 Description . . . 347
6.4.1.2 Application to finance . . . 347
6.4.2 Time-varying feedforward networks . . . 348

7 Towards dynamically stable reservoirs 350
7.1 A specific implementation . . . 350
7.1.1 Overview of the topology . . . 350
7.1.2 Cost function . . . 351
7.1.3 Random generation of the input and feedback matrices . . . 351
7.1.4 Topology of the reservoir matrix . . . 351
7.1.5 Reservoir dynamics . . . 352
7.2 Optimisation methods . . . 353
7.2.1 Random Generation: GEN . . . 355
7.2.2 Stochastic Gradient Descent: SGD . . . 356
7.3 Adaptive filters for online learning . . . 358
7.3.1 A review of adaptive filtering theory . . . 358
7.3.1.1 Simple gradient descent . . . 360
7.3.1.2 Recursive least squares . . . 360

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

7.3.2 Online algorithm . . . 361
7.4 Some results . . . 361
7.4.1 Using Offline algorithms . . . 361
7.4.1.1 The Mackey-Glass series . . . 361
7.4.1.2 Financial time series . . . 363
7.4.2 Using online algorithms . . . 364
7.4.2.1 The Mackey-Glass series . . . 364
7.4.2.2 Financial time series . . . 367
7.5 Associative Reservoir Computing . . . 369
7.5.1 Paradigm . . . 369
7.5.1.1 Definitions . . . 369
7.5.1.2 The model . . . 369
7.5.1.3 Attractor-based computation . . . 370
7.5.2 States stabilisation techniques . . . 371
7.5.2.1 Reservoir dynamics calibration to observable stable states . . . 372
7.5.2.2 Learning with filtered inputs . . . 372

8 Introduction to unsupervised learning 374
8.1 Batch intrinsic plasticity . . . 374
8.1.1 Description . . . 374
8.1.2 Some results . . . 377
8.2 Radial basis function . . . 378
8.2.1 Description . . . 378
8.2.2 Some results . . . 381
8.3 Recurrent RBF network . . . 386
8.3.1 Standard RRBFN . . . 386
8.3.1.1 Pseudo code . . . 386
8.3.1.2 Parameter tuning . . . 387
8.3.2 Randomly parametrised RRBFN . . . 388
8.3.3 Directional parametrised RRBFN . . . 389
8.3.4 Empirical results . . . 389
8.3.5 Unsupervised learning for RRBFN . . . 390
8.3.5.1 Pseudo code . . . 392
8.3.5.2 Empirical results . . . 393

9 Deep learning 399
9.1 Some deep neural networks . . . 400
9.1.1 Convolutional neural network . . . 400
9.1.1.1 An overview . . . 400
9.1.1.1.1 Description . . . 400
9.1.1.1.2 Some history . . . 401
9.1.1.2 Modelling images . . . 403
9.1.1.2.1 Biological nervous systems . . . 403
9.1.1.2.2 The convolution operation . . . 403
9.1.1.3 The CNN architecture . . . 403
9.1.1.3.1 Convolutional layers . . . 403
9.1.1.3.2 Pooling . . . 404
9.1.1.3.3 Receptive field . . . 405
9.1.1.3.4 Weights . . . 406
9.1.1.3.5 Spatial arrangement . . . 406

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

9.1.1.4 Mathematical representation . . . 408
9.1.1.4.1 Convolutional layer . . . 408
9.1.1.4.2 Non-linearity layer . . . 410
9.1.1.4.3 Rectification . . . 410
9.1.1.4.4 Feature pooling and sub-sampling layer . . . 410
9.1.1.4.5 Fully connected layer . . . 410
9.2 Training deep networks . . . 410
9.2.1 Highway networks . . . 411
9.2.1.1 Description . . . 411
9.2.1.2 Implementation . . . 412
9.2.1.3 Shortcomings . . . 412

III The mathematical formalism 413

10 Supervised learning 414
10.1 Introduction to learning theory . . . 414
10.1.1 Function estimation . . . 414
10.1.1.1 Risk minimisation . . . 414
10.1.1.2 Empirical risk minimisation . . . 415
10.1.1.2.1 Convergence . . . 416
10.1.1.2.2 Structural risk minimisation . . . 416
10.1.1.3 Model selection . . . 417
10.1.1.3.1 Bonds for neural networks . . . 417
10.1.1.3.2 Bias-variance tradeoff . . . 418
10.1.1.3.3 Overfitting . . . 420
10.1.2 Solving the optimisation problem . . . 421
10.1.2.1 The hypothesis function . . . 421
10.1.2.2 The loss function . . . 423
10.1.2.3 Regularisation and stability . . . 425
10.1.2.3.1 Early stopping . . . 425
10.1.2.3.2 Batch normalisation . . . 426
10.1.2.3.3 Dropout . . . 426
10.1.2.4 Solving for the optimum . . . 426
10.1.2.4.1 The batch gradient descent . . . 427
10.1.2.4.2 The stochastic gradient descent . . . 427
10.1.2.5 Online convex optimisation . . . 428
10.1.3 Methods for stochastic optimisation . . . 428
10.1.3.1 Settings . . . 429
10.1.3.2 The Adam algorithm . . . 429
10.1.3.3 The AdaGrad algorithm . . . 430
10.1.4 Solving the loss function . . . 430
10.1.4.1 Data taking continuous values . . . 430
10.1.4.2 Gaussian distribution . . . 431
10.2 Dynamical systems . . . 432
10.2.1 Mathematical formulation . . . 432
10.2.2 Handling memory . . . 434

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

11 Statistical models **436**
11.1 Introduction . . . 436
11.1.1 Terminology . . . 436
11.1.2 Basic tools for summarising and forecasting data . . . 438
11.1.2.1 Presenting forecasting methods . . . 438
11.1.2.2 Summarising the data . . . 439
11.1.2.3 Measuring the forecasting accuracy . . . 443
11.1.2.4 Prediction intervals . . . 445
11.1.2.5 Estimating model parameters . . . 446
11.2 Statistical inference . . . 446
11.2.1 Modelling time series . . . 447
11.2.1.1 Structural time series . . . 447
11.2.1.2 Some simple statistical models . . . 447
11.2.1.2.1 AR process . . . 447
11.2.1.2.2 MA process . . . 448
11.2.1.2.3 ARMA process . . . 448
11.2.1.2.4 ARIMA process . . . 448
11.2.1.2.5 ARCH process . . . 448
11.2.1.2.6 GARCH process . . . 449
11.2.1.2.7 Example of a financial model . . . 449
11.2.2 Introducing parametric regression . . . 449
11.2.2.1 Some rules for conducting inference . . . 450
11.2.2.1.1 Frequentist inference . . . 450
11.2.2.1.2 Bayesian inference . . . 450
11.2.2.1.3 Differences . . . 450
11.2.2.2 The least squares estimator . . . 451
11.2.2.2.1 Maximum likelihood estimator . . . 451
11.2.2.2.2 Linear models . . . 452
11.2.2.2.3 Nonlinear models . . . 453
11.2.3 Introducing Bayesian statistics . . . 453
11.2.3.1 Bayesian inference . . . 453
11.2.3.2 Marginal likelihood function . . . 455
11.2.3.3 Variational Bayesian methods . . . 456
11.2.4 Statistical classification . . . 457
11.2.4.1 Description . . . 457
11.2.4.2 The generative and discriminative approaches . . . 458
11.2.4.3 Relationship between models . . . 459
11.2.4.4 Classifying the data . . . 459
11.2.4.5 Types of models . . . 460
11.2.4.5.1 Generative models . . . 460
11.2.4.5.2 Discriminative models . . . 460
11.2.5 Introducing state-space models . . . 461
11.2.5.1 The state-space form . . . 461
11.2.5.1.1 The inference mechanism . . . 461
11.2.5.1.2 Some problems . . . 462
11.2.5.2 The Kalman filter . . . 462
11.2.5.2.1 Parameter estimation . . . 463
11.2.5.2.2 Smoothing . . . 464
11.2.5.3 Model specification . . . 464
11.3 SVM models . . . 465

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.3.1 An optimisation problem . . . 465
11.3.2 The algorithms . . . 467
11.3.2.1 Online algorithms . . . 468
11.3.2.1.1 A simple algorithm . . . 468
11.3.2.1.2 More complex algorithms . . . 468
11.3.2.1.3 Parallel algorithm . . . 469
11.3.2.2 Offline algorithms . . . 469
11.3.2.2.1 The Lagrange multiplier . . . 469
11.3.2.2.2 The kernel trick . . . 470
11.3.3 Reproducing kernel Hilbert spaces . . . 472
11.3.3.1 The Hilbert spaces . . . 472
11.3.3.2 Nonparametric regression . . . 473
11.4 Some discriminative models . . . 473
11.4.1 Mixture density network . . . 473
11.4.1.1 Mixture models . . . 473
11.4.1.2 The MDN model . . . 475
11.4.1.3 Summary . . . 477
11.4.1.4 Improvements . . . 478
11.5 Probabilistic topic models . . . 478
11.5.1 Overview . . . 479
11.5.1.1 Generative models . . . 479
11.5.1.2 Notation . . . 480
11.5.1.3 Geometric interpretation . . . 481
11.5.2 Simple latent variable models . . . 481
11.5.2.1 The unigram model . . . 481
11.5.2.2 The mixture of unigrams model . . . 481
11.5.2.3 The probabilistic latent semantic indexing model . . . 482
11.5.3 Latent Dirichlet allocation . . . 483
11.5.3.1 The algorithm . . . 483
11.5.3.2 Description . . . 483
11.5.3.3 A variant . . . 485

12 Dynamical models **487**
12.1 Forecasting models . . . 487
12.1.1 Regression models . . . 487
12.1.1.1 Data normalisation . . . 487
12.1.1.2 Description . . . 488
12.1.1.3 Linear regression . . . 489
12.1.2 Generalised nonlinear nonparametric models . . . 490
12.1.2.1 Presentation . . . 490
12.1.2.2 Describing the models . . . 490
12.1.3 Example: forecasting volatility . . . 492
12.1.3.1 Presentation . . . 492
12.1.3.2 A first decomposition . . . 493
12.1.3.3 The structure of volatility models . . . 493
12.1.3.3.1 Presentation . . . 493
12.1.3.3.2 Benchmark volatility models . . . 494
12.2 Introduction to HMM . . . 495
12.2.1 Description . . . 496
12.2.1.1 The model . . . 496

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

12.2.1.2 Maximum likelihood revisited . . . 499
12.2.1.2.1 Complete data . . . 499
12.2.1.2.2 Estimation with hidden variables . . . 499
12.2.2 The forward-backward algorithm . . . 500
12.2.2.1 The forward algorithm . . . 500
12.2.2.2 The forward-backward algorithm . . . 501
12.2.3 The Viterbi algorithm . . . 503
12.2.4 Estimating parameters: The Baum-Welch algorithm . . . 504
12.3 Some generative models . . . 505
12.3.1 Overview . . . 505
12.3.2 Autoencoding variational Bayes . . . 506
12.3.2.1 Description . . . 507
12.3.2.1.1 The problem . . . 507
12.3.2.1.2 The solution . . . 508
12.3.2.2 The SGVB . . . 508
12.3.2.3 The reparametrisation . . . 509
12.3.2.4 The variational AE . . . 510
12.3.3 Generative adversarial networks . . . 510
12.3.3.1 Description . . . 510
12.3.3.2 Analysis . . . 513
12.3.3.3 Improvements . . . 514
12.3.3.4 Conditional generative adversarial networks . . . 515
12.4 A review of anomaly detection . . . 515
12.4.1 Feature extraction . . . 515
12.4.2 Learning feature representations of normality . . . 516
12.4.2.1 Autoencoders . . . 516
12.4.2.2 Generative adversarial networks . . . 518
12.4.2.3 Predictability modelling . . . 520
12.4.2.4 Self-supervised classification . . . 520
12.4.3 End-to-end one-class classification . . . 521
12.5 Introduction to complex dynamical systems . . . 523
12.5.1 Overview . . . 523
12.5.2 Deep generative Markov state models . . . 523
12.6 Modelling sequential data . . . 526
12.6.1 RNNs as statistical models . . . 526
12.6.1.1 An Overview . . . 526
12.6.1.2 RNNs as state-space models . . . 527
12.6.1.3 The particle filter . . . 528
12.6.1.4 Link with generative models . . . 529
12.6.2 Generating sequences . . . 531
12.6.2.1 Description . . . 531
12.6.2.2 Generating sequences with RNNs . . . 532
12.6.2.2.1 An overview . . . 532
12.6.2.2.2 The stacked LSTM model . . . 532
12.6.2.2.3 Mathematical representation . . . 534
12.6.3 Reconstructing and predicting sequences . . . 535
12.6.3.1 The encoder-decoder-predictor model . . . 535
12.6.3.1.1 The autoencoder model . . . 535
12.6.3.1.2 The predictor model . . . 536
12.6.3.1.3 The composite model . . . 536

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

12.6.3.2 Convolutional LSTM network . . . 537
12.6.3.2.1 Mathematical formulation . . . 537
12.6.3.2.2 The model . . . 538

13 Graph networks **541**
13.1 Graph theory . . . 541
13.1.1 Description . . . 541
13.1.1.1 An overview . . . 541
13.1.1.2 Graph concepts . . . 542
13.1.1.2.1 Some concepts . . . 543
13.1.1.2.2 Distance . . . 544
13.1.1.2.3 Clique . . . 545
13.1.1.3 Storage . . . 545
13.1.1.4 Hierarchical structure . . . 545
13.1.1.5 Some simple graphs . . . 546
13.1.1.5.1 Regular graphs . . . 546
13.1.1.5.2 Complete graphs . . . 546
13.1.2 Examples of combinatorial problems . . . 547
13.1.2.1 Shortest path problems . . . 547
13.1.2.1.1 Description . . . 547
13.1.2.1.2 Bellman's optimality principle . . . 547
13.1.2.1.3 Shortest spanning trees . . . 548
13.1.2.2 Networks . . . 548
13.1.2.2.1 Flow augmenting paths . . . 548
13.1.2.2.2 Cut sets . . . 549
13.1.3 Directed acyclic graph . . . 549
13.1.3.1 Some properties . . . 550
13.1.4 Topological ordering and sorting . . . 552
13.1.4.1 Topological ordering . . . 552
13.1.4.2 Topological sorting . . . 552
13.1.5 Causal structures . . . 553
13.1.6 Some applications . . . 553
13.1.7 Random graphs . . . 554
13.1.7.1 Overview . . . 554
13.1.7.2 Description . . . 555
13.1.7.2.1 The Erdos-Renyi model . . . 555
13.1.7.3 Some properties . . . 556
13.1.7.4 Some measures of topology . . . 558
13.1.7.4.1 Degree distribution . . . 558
13.1.7.4.2 Diameter . . . 558
13.1.7.4.3 Clustering coefficient . . . 559
13.1.7.4.4 Graph spectra . . . 559
13.2 Graphical models . . . 560
13.2.1 Description . . . 560
13.2.2 Bayesian networks . . . 561
13.2.2.1 Description . . . 561
13.2.2.2 Some concepts . . . 561
13.2.2.2.1 Factorisation . . . 561
13.2.2.2.2 The local Markov property . . . 563
13.2.2.2.3 The Markov blanket . . . 563

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

13.2.2.2.4 Causal networks . . . 563
13.2.2.2.5 Implementation . . . 564
13.2.2.2.6 Approximation algorithms . . . 564
13.2.2.3 Inference tasks . . . 564
13.3 Complex networks . . . 565
13.3.1 Generalised random graphs . . . 566
13.3.1.1 Measures of topology . . . 566
13.3.1.1.1 Clustering coefficient . . . 566
13.3.1.1.2 The degree distribution . . . 566
13.3.1.2 Social networks . . . 567
13.3.1.3 Building blocks . . . 567
13.3.1.3.1 Ring lattices . . . 567
13.3.1.3.2 Random graphs . . . 568
13.3.1.3.3 A small world network . . . 569
13.3.2 Small-world networks . . . 569
13.3.2.1 Overview . . . 569
13.3.2.2 Description . . . 569
13.3.3 Scale-free networks . . . 570
13.3.3.1 Description . . . 570
13.3.3.1.1 Overview . . . 570
13.3.3.1.2 Some properties . . . 571
13.3.3.2 Building networks . . . 571
13.3.3.3 Properties of the BA-model . . . 572
13.3.3.4 Evolving networks . . . 573
13.3.4 Fractal networks . . . 575
13.3.4.1 Overview . . . 575
13.3.4.2 Methods for computing the dimension . . . 576
13.3.4.2.1 The box counting method . . . 576
13.3.4.2.2 The cluster growing method . . . 576
13.3.4.2.3 Box counting and renormalisation . . . 576
13.3.4.2.4 Skeleton and fractal scaling . . . 577
13.3.5 Multifractal networks . . . 577
13.3.5.1 The sandbox algorithm . . . 578
13.3.5.1.1 The fixed size box-counting algorithm . . . 578
13.3.5.1.2 Characterising multifractality . . . 579
13.3.5.1.3 The algorithm . . . 579

14 Neural NLP **581**
14.1 Sequence to sequence . . . 581
14.1.1 Description . . . 581
14.1.1.1 Overview . . . 581
14.1.1.1.1 Seq2Seq model . . . 581
14.1.1.1.2 The encoder and decoder . . . 582
14.1.1.2 Attention mechanisms . . . 584
14.1.1.2.1 Description . . . 584
14.1.1.2.2 Scoring the states . . . 586
14.1.1.3 Some simplified attentions . . . 587
14.1.1.3.1 A simple and improved attention . . . 587
14.1.1.3.2 A hierarchical attention network . . . 589
14.1.1.3.3 Implementation . . . 590

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

14.1.1.3.4 Application to language translator . . . 592
14.1.2 Modelling Seq2Seq . . . 593
14.1.2.1 Introduction . . . 593
14.1.2.1.1 A brief history . . . 593
14.1.2.1.2 Settings . . . 594
14.1.2.2 The recurrent language model . . . 594
14.1.2.3 The encoder-decoder process . . . 595
14.1.2.3.1 The RNN encoder-decoder . . . 596
14.1.2.3.2 The RNNsearch . . . 598
14.1.2.4 The Transformer . . . 599
14.2 Solving NLP tasks with CNNs . . . 601
14.2.1 Text classification problems . . . 601
14.2.1.1 Overview . . . 601
14.2.1.2 Using tensor products . . . 602
14.3 Transfer learning . . . 605
14.3.1 Pre-training in NLP . . . 605
14.4 Description of some models . . . 607
14.4.1 OpenAI GPT . . . 607
14.4.2 BERT . . . 608
14.4.3 Masked sequence to sequence pre-training . . . 608
14.5 Sentence pair modelling . . . 610
14.5.1 Introduction . . . 610
14.5.2 Sentence embeddings . . . 611
14.5.2.1 Sentence encoder models . . . 612
14.5.2.2 Sentence-BERT . . . 614
14.5.3 String matching . . . 615
14.5.3.1 Mathematical formulation . . . 616
14.5.3.2 Base string encoder . . . 617
14.5.3.3 Aggregating RNN states . . . 618
14.5.3.4 Hierarchical BiRNN with max-pooling . . . 620
14.5.3.5 Using common features . . . 620
14.5.3.6 Architecture for toponym matching . . . 621
14.5.4 Semantic search . . . 621

IV Stochastic Control 622

15 Markov Decision Problems 623
15.1 Presentation . . . 623
15.1.1 Decision-making problems . . . 623
15.1.2 Description . . . 624
15.2 Learning with stationary reward . . . 627
15.2.1 Description . . . 627
15.2.2 The REINFORCE algorithm . . . 628
15.2.2.1 Some examples . . . 629
15.2.2.1.1 Bernoulli neurons . . . 629
15.2.2.1.2 Binary LN neurons . . . 629
15.3 Markov decision process . . . 630
15.3.1 The problem . . . 631
15.3.1.1 Description . . . 631

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

15.3.1.2 The transition probabilities . . . 633
15.3.1.3 Performance metrics . . . 635
15.3.1.4 The value function . . . 636
15.3.2 The Bellman equations . . . 637
15.3.2.1 Description . . . 637
15.3.2.2 Approximation . . . 638
15.3.3 A mathematical formalism . . . 639
15.3.3.1 Overview . . . 639
15.3.3.1.1 Formulation of the problem . . . 639
15.3.3.1.2 Solving the problem . . . 641
15.3.3.2 The value function . . . 642
15.3.3.2.1 The optimal policy . . . 643
15.3.3.3 The Q-value function . . . 644
15.3.4 Approximation . . . 648
15.3.4.1 Known transition function and reward . . . 648
15.3.4.2 Unknown transition function and reward . . . 650
15.3.4.3 Stochastic approximation . . . 650
15.3.4.3.1 Description . . . 651
15.4 Introduction to stochastic control . . . 652
15.4.1 Description of the problem . . . 652
15.4.2 HJB: the Markov control . . . 653
15.4.2.1 Description . . . 653
15.4.2.2 Some examples . . . 655
15.4.3 Terminal conditions . . . 656
15.4.4 Dynamic programming . . . 656
15.4.4.1 Dynamic programming principle . . . 657
15.4.4.2 Dynamic programming equation . . . 657

16 Solving conditional expectations 659
16.1 Temporal difference learning . . . 659
16.1.1 Introduction . . . 659
16.1.1.1 Overview . . . 659
16.1.1.2 Model assumptions . . . 661
16.1.2 Temporal credit assignment . . . 661
16.1.2.1 Nonassociative task . . . 661
16.1.2.2 Associative task . . . 663
16.1.2.3 Delayed reinforcement . . . 664
16.1.2.3.1 Primary reinforcement . . . 664
16.1.2.3.2 Neuronlike adaptive elements . . . 665
16.1.2.4 The AHC-algorithm . . . 667
16.1.2.4.1 Description . . . 668
16.1.2.4.2 Link to other models . . . 670
16.1.2.4.3 A generalisation . . . 671
16.1.3 The TD family of learning procedures . . . 671
16.1.3.1 The $TD(1)$ procedure . . . 672
16.1.3.2 The $TD(\lambda)$ procedure . . . 673
16.1.3.3 A random-walk example . . . 673
16.1.3.4 Convergence . . . 677
16.1.3.5 TD as a gradient descent . . . 678
16.1.3.6 Intra-sequence weight update . . . 679

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

16.1.3.7 Predicting cumulative outcomes . . . 680
16.1.3.7.1 Finite predictions . . . 680
16.1.3.7.2 Infinite discounted predictions . . . 680
16.1.4 TD revisited and Q-learning . . . 681
16.1.4.1 Estimating expected returns . . . 681
16.1.4.2 Estimating and learning the action-value function . . . 684
16.1.4.2.1 Estimating the Q-function . . . 684
16.1.4.2.2 Learning the Q-function . . . 685
16.1.4.2.3 Convergence of one-step Q-learning . . . 686
16.1.5 Finite time performance . . . 687
16.1.5.1 Overview . . . 687
16.1.5.2 Adaptive learning rate . . . 687
16.2 Combining supervised learning and TD procedures . . . 689
16.2.1 Overview . . . 689
16.2.2 Multilayer connectionist system . . . 690
16.2.2.0.1 The evaluation network . . . 691
16.2.2.0.2 The action network . . . 692
16.2.3 Implementing the TDBP model . . . 693
16.2.3.1 Feedforward networks . . . 693
16.2.3.2 Implementing the TD-procedures . . . 694
16.2.3.2.1 The network . . . 695
16.2.3.2.2 The $TD(0)$ case . . . 695
16.2.3.3 The TDBP model . . . 696
16.3 Beyond TD-learning . . . 698
16.3.1 State representation . . . 698
16.3.1.1 Predictive state representation . . . 698
16.3.1.1.1 Description . . . 698
16.3.1.1.2 The TD networks . . . 700
16.3.1.1.3 The Bellman network equation . . . 701
16.3.1.2 Gradient temporal difference networks . . . 702
16.3.2 A link to forward start contract . . . 703
16.4 Deep BSDE methods . . . 705
16.4.1 Preliminary . . . 705
16.4.1.1 Financial contracts . . . 705
16.4.2 European options in the constrained case . . . 706
16.4.3 Introducing the FBSDE . . . 708
16.4.4 Deep BSDE . . . 709
16.4.4.1 Description . . . 709
16.4.4.2 Drawbacks . . . 711
16.4.5 Forward-backward stochastic neural networks . . . 711
16.4.5.1 Description . . . 711
16.4.5.2 Link to the TD-procedures . . . 712
Annexes . . . 714
16.5 Classical conditioning . . . 714
16.5.1 Overview . . . 714
16.5.2 Some conditioning . . . 714
16.5.2.1 Forward conditioning . . . 714
16.5.2.2 Second-order or higher-order conditioning . . . 714
16.5.2.3 Backward conditioning . . . 715
16.5.2.4 Temporal conditioning . . . 715

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

16.5.2.5 Extinction procedure . . . 715
16.5.3 The Rescorla-Wagner model . . . 715

17 Reinforcement learning **716**
17.1 Learning the optimal policy . . . 716
17.1.1 An overview . . . 716
17.1.1.1 The RL problem . . . 716
17.1.1.2 Types of RL algorithms . . . 717
17.1.1.2.1 Model-based versus model-free . . . 717
17.1.1.2.2 Some examples . . . 717
17.1.1.2.3 Off-policy versus on-policy . . . 718
17.1.1.3 Policy optimisation . . . 718
17.1.1.3.1 The algorithms . . . 718
17.1.1.3.2 Comparison . . . 719
17.1.1.3.3 Implementation . . . 719
17.1.1.4 Generalisation . . . 719
17.1.2 The policy iteration . . . 720
17.1.2.1 Introduction . . . 720
17.1.2.2 Value Iteration . . . 721
17.1.2.3 Monte Carlo method . . . 722
17.1.2.4 TD learning . . . 722
17.1.2.4.1 Description . . . 722
17.1.2.4.2 Model assumptions . . . 723
17.1.2.4.3 The SARSA model . . . 723
17.1.2.5 Fitted value iteration . . . 724
17.1.2.5.1 The value function . . . 724
17.1.2.5.2 The Q-value function . . . 724
17.1.3 The Q-learning algorithms . . . 725
17.1.3.1 Q-learning . . . 725
17.1.3.2 A generalisation . . . 726
17.1.3.2.1 The AHCON . . . 727
17.1.3.2.2 The QCON . . . 728
17.1.3.2.3 Experience replay . . . 729
17.1.3.2.4 Using $TD(\lambda)$ . . . 730
17.1.3.2.5 The modified QCON . . . 732
17.1.3.3 Neural fitted Q-value . . . 732
17.1.3.4 Deep Q-learning . . . 732
17.1.3.5 Experience replay . . . 734
17.1.3.6 Convergence . . . 735
17.1.3.6.1 The problem . . . 735
17.1.3.6.2 The double Q-learning . . . 736
17.1.3.6.3 The multi-step returns . . . 736
17.1.3.7 The continuous state-action method . . . 736
17.1.3.7.1 The continuous action method . . . 737
17.1.3.7.2 The continuous state-action method . . . 738
17.1.4 The policy gradient methods . . . 738
17.1.4.1 The REINFORCE algorithm . . . 739
17.1.4.1.1 Example . . . 742
17.1.4.2 Variance reduction . . . 742
17.1.4.2.1 Some solutions . . . 742

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.1.4.2.2 Baseline function . . . 743
17.1.5 The Actor-Critic algorithms . . . 743
17.1.5.1 Policy gradient methods with approximator . . . 744
17.1.5.1.1 Description . . . 744
17.1.5.1.2 Variance reduction . . . 746
17.1.5.1.3 The temporal-difference method . . . 747
17.1.5.2 Description . . . 748
17.1.5.2.1 Overview . . . 749
17.1.5.2.2 The value function . . . 750
17.1.5.2.3 The Q-value function . . . 750
17.1.5.2.4 The main steps . . . 751
17.1.5.3 The algorithm . . . 751
17.1.6 Gradient base actor-critic . . . 753
17.1.6.1 State-dependent baselines . . . 753
17.1.6.2 Continuous action methods . . . 753
17.1.6.2.1 The deterministic policy gradient . . . 753
17.1.6.2.2 Extensions . . . 755
17.1.7 Policy gradient improvement . . . 756
17.1.7.1 Policy iteration . . . 756
17.1.7.2 An optimisation problem . . . 758
17.1.7.2.1 Recovering the natural policy gradient . . . 759
17.2 Minimising risk in MDP . . . 760
17.2.1 Measures of risk . . . 760
17.2.2 The variance-penalised MDP . . . 761
17.2.2.1 Bellman equations . . . 762
17.2.2.2 The known model . . . 762
17.2.2.3 The unknown model . . . 764
17.3 Continuous state spaces . . . 765

V Optimisation **766**

18 Introduction to constrained optimisation **767**
18.1 Introduction . . . 767
18.1.1 Defining the problem . . . 767
18.1.1.1 The constrained problem . . . 767
18.1.1.2 Some examples . . . 768
18.1.2 Search techniques . . . 769
18.1.3 Discrete optimisation . . . 771
18.1.3.1 Defining the TSP . . . 771
18.1.3.2 Some solution methods . . . 773
18.1.3.2.1 Exact methods . . . 773
18.1.3.2.2 Heuristic methods . . . 773
18.2 Conditions on optima . . . 774
18.2.1 Conditions for a local optimum . . . 775
18.2.1.1 Local minimum . . . 775
18.2.1.2 Local maximum . . . 775
18.2.2 Conditions for constrained local optimum . . . 775
18.2.2.1 The equality constraint . . . 775
18.2.2.2 The inequality constraint . . . 776

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

18.2.2.3 A summary . . . 776
18.2.3 Conditions for a global optimum . . . 776
18.2.4 Some solutions . . . 777
18.3 Unconstrained optimisation . . . 778
18.3.1 Some classical methods . . . 778
18.3.1.1 The gradient descent . . . 778
18.3.1.2 The subgradient descent . . . 778
18.3.1.3 The Newton's method . . . 779
18.3.1.4 The convergence . . . 780
18.3.2 Adding constraints . . . 780
18.3.2.1 The projected subgradient descent . . . 780
18.3.2.2 The subgradient descent . . . 780
18.3.2.3 The Newton method . . . 781
18.4 A first approach to constrained optimisation . . . 781
18.4.1 Penalty and barriers . . . 781
18.4.1.1 Log barrier method . . . 782
18.4.1.2 Central path . . . 782
18.4.1.3 Squared penalty method . . . 782
18.4.2 The augmented Lagrangian approach . . . 783
18.4.2.1 The equality constraint . . . 783
18.4.2.2 The inequality constraint . . . 784
18.5 The Lagrangian approach . . . 784
18.5.1 The KKT conditions . . . 784
18.5.2 Implication from the KKT conditions . . . 786
18.5.2.1 Equality constraints . . . 786
18.5.2.2 Inequality constraints . . . 787
18.5.3 The Lagrangian dual problem . . . 787
18.5.3.1 Definition . . . 787
18.5.3.2 Relation between primal and dual solutions . . . 788
18.5.3.3 Interpretation . . . 788
18.5.3.4 Summary . . . 789
18.5.3.5 Equivalence . . . 790
18.5.4 Additional . . . 791
18.5.4.1 The log barrier method revisited . . . 791
18.5.4.2 Non-convex optimisation . . . 791
18.6 Standard convex programs . . . 792
18.6.1 Regularisation . . . 792
18.6.2 Simple programming models . . . 792
18.6.2.1 Quadratic programming . . . 792
18.6.2.2 Linear programming . . . 793
18.6.2.2.1 Linear programming . . . 793
18.6.2.2.2 Integer linear programming . . . 793
18.6.2.3 Second-order cone programming . . . 794
18.6.2.4 Semidefinite programming . . . 794
18.6.3 Some examples . . . 794
18.6.3.1 Quadratic programming . . . 794
18.6.3.2 The Lasso problem . . . 794
18.6.3.3 ILP formulations . . . 795
18.6.3.3.1 $(0,1)$-Knapsack . . . 795
18.6.3.3.2 Assignment problem . . . 795

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

18.6.3.3.3 ATSP . . . 795
18.7 Generalising the penalty function approach . . . 796
18.7.1 The energy function . . . 796
18.7.2 The penalty functions . . . 797
18.7.2.1 Type I . . . 797
18.7.2.2 Type II . . . 798
18.7.2.3 Type III . . . 798

19 Global search optimisation **800**
19.1 Evolutionary algorithms . . . 800
19.1.1 Description . . . 800
19.1.1.1 A brief history of evolutionary algorithms . . . 800
19.1.1.2 Some terminology . . . 801
19.1.1.3 Adaptation to the environment . . . 802
19.1.2 Some optimisation methods . . . 806
19.1.2.1 Random optimisation . . . 806
19.1.2.2 Particle swarm optimisation . . . 807
19.1.2.3 Cross entropy optimisation . . . 809
19.1.2.4 Simulated annealing . . . 810
19.1.3 Introduction to genetic algorithms . . . 811
19.1.3.1 Description . . . 811
19.1.3.2 Genetic operators . . . 812
19.1.3.2.1 Selection . . . 813
19.1.3.2.2 Recombination . . . 813
19.1.3.2.3 Mutation . . . 814
19.1.3.2.4 Replacement . . . 814
19.1.3.3 Schema theories . . . 815
19.1.3.4 Dual representation and interpretation function . . . 815
19.1.3.5 Fitness landscapes . . . 816
19.1.3.6 Multimodal functions . . . 816
19.1.3.6.1 Sharing . . . 817
19.1.3.7 Solving the TSP . . . 818
19.1.3.8 Advantages and disadvantages . . . 820
19.1.4 Introduction to differential evolution . . . 820
19.1.4.1 The DE algorithm . . . 820
19.1.4.1.1 The mutation . . . 821
19.1.4.1.2 The recombination . . . 821
19.1.4.1.3 The selection . . . 822
19.1.4.1.4 Simple convergence criteria . . . 822
19.1.4.2 Pseudocode . . . 823
19.1.4.3 The strategies . . . 824
19.1.4.3.1 Scheme DE1 . . . 824
19.1.4.3.2 Scheme DE2 . . . 824
19.1.4.3.3 Scheme DE3 . . . 824
19.1.4.3.4 Scheme DE4 . . . 824
19.1.4.3.5 Scheme DE5 . . . 825
19.1.4.3.6 Scheme DE6 . . . 826
19.1.4.3.7 Scheme DE7 . . . 826
19.1.4.3.8 Scheme DE8 . . . 826
19.1.4.4 Improvements . . . 827

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

19.1.4.4.1 The tuning parameters . . . 827
19.1.4.4.2 Ageing . . . 827
19.1.4.4.3 Constraints on parameters . . . 828
19.1.4.4.4 Convergence . . . 828
19.1.4.4.5 Self-adaptive parameters . . . 828
19.1.4.4.6 Selection . . . 829
19.1.4.5 Convergence criteria revised . . . 830
19.2 Handling the constraints . . . 832
19.2.1 Describing the problem . . . 832
19.2.2 Defining the feasibility rules . . . 833
19.2.3 Improving the feasibility rules . . . 834
19.2.4 Handling diversity . . . 835
19.2.5 The proposed algorithm . . . 836
19.3 Genetic programming . . . 837
19.3.1 An overview . . . 837
19.3.2 Description . . . 838
19.3.2.1 Representation, initialisation and operators . . . 839
19.3.2.1.1 Representation . . . 839
19.3.2.1.2 Initialisation . . . 840
19.3.2.1.3 Operators . . . 840
19.3.2.2 Selection, recombination and mutation . . . 840
19.3.2.2.1 Selection . . . 840
19.3.2.2.2 Crossover . . . 841
19.3.2.2.3 Mutation . . . 842
19.3.3 Running a GP . . . 842
19.3.3.1 The terminal set . . . 843
19.3.3.2 The function set . . . 843
19.3.3.3 The fitness measure . . . 844
19.3.3.4 The parameters and termination . . . 846
19.3.4 Advanced techniques . . . 846
19.3.4.1 Abstraction . . . 846
19.3.4.1.1 Automatically defined functions . . . 847
19.3.4.1.2 Abstract data types . . . 847
19.3.4.2 Breeding policies . . . 847
19.3.4.2.1 Population enrichment . . . 847
19.3.4.2.2 Tournament selection . . . 847
19.3.4.2.3 Steady state GP . . . 847
19.3.4.2.4 Elitism . . . 847
19.3.4.2.5 Demes . . . 848
19.3.4.3 Program architecture . . . 848
19.3.4.4 Problem solver . . . 848
19.3.4.5 Constraining syntactic structures . . . 848
19.3.4.5.1 Enforcing particular structures . . . 849
19.3.4.5.2 Strongly typed GP . . . 849
19.3.4.5.3 Grammar based constraints . . . 849
19.3.4.5.4 Some limitations . . . 850
19.3.4.6 Fitness evaluation . . . 850
19.3.4.6.1 Absolute and relative fitness . . . 850
19.3.4.6.2 Multi-phase fitness . . . 850
19.3.4.6.3 Full and partial fitness . . . 851

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

19.3.4.6.4 Evaluation efficiency . . . 851
19.3.4.6.5 Pareto scoring . . . 851
19.3.4.7 Other genetic operators . . . 851
19.3.4.8 Developmental GP . . . 852
19.3.4.9 Read's code . . . 852
19.3.4.10 Some improvements . . . 853
19.3.4.11 Fitness landscapes . . . 854
19.3.5 Other types of GP . . . 854
19.3.5.1 Linear GP . . . 855
19.3.5.2 Graph based GP . . . 856
19.3.6 Some applications . . . 857
19.3.6.1 Symbolic regression . . . 857
19.3.6.2 Image and signal processing . . . 858
19.3.6.3 Financial trading . . . 858
19.3.7 General comments . . . 859
19.3.7.1 Fitness with caches . . . 859
19.3.7.2 Some recommendations . . . 859
19.4 Multi-objective optimisation problems . . . 860
19.4.0.1 Defining truly multi-objective problems . . . 861
19.4.0.2 Using genetic algorithms . . . 862

20 Evolving networks **863**
20.1 Introduction to neuroevolution . . . 864
20.1.1 The framework . . . 864
20.1.1.1 Types of topology . . . 864
20.1.1.2 Genome encoding . . . 865
20.1.1.3 Crossover . . . 866
20.1.1.4 Basic algorithm . . . 868
20.1.1.5 Some algorithms . . . 868
20.1.2 Improving the search . . . 869
20.1.2.1 Diversity . . . 869
20.1.2.2 Novelty and quality diversity . . . 870
20.1.2.3 Network shaping . . . 870
20.1.2.4 Bootstrapping . . . 871
20.1.3 Large networks . . . 872
20.1.3.1 Compressing patterns . . . 872
20.1.3.2 Structure reuse . . . 872
20.1.3.3 Incremental complexification . . . 872
20.1.3.4 Search space restriction . . . 872
20.1.4 NEAT algorithm . . . 873
20.1.4.1 Genetic encoding . . . 873
20.1.4.2 Crossover . . . 874
20.1.4.3 Speciation . . . 876
20.1.4.4 Complexification . . . 877
20.2 Hybrid intelligent modelling . . . 877
20.2.1 ANN plus GA in finance . . . 877
20.2.1.1 An overview . . . 877
20.2.1.2 Some examples . . . 878
20.2.1.2.1 Feature selection . . . 878
20.2.1.2.2 Weight selection . . . 878

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

20.2.2 ANN plus DE in finance . . . 879

21 Solving optimisation problems with ML **881**
21.1 Solving constrained optimisation problems . . . 881
21.1.1 The method . . . 881
21.1.1.1 A short review . . . 881
21.1.1.2 Mapping optimisation problems to networks . . . 882
21.1.2 Some optimisation models . . . 882
21.1.2.1 Discrete synchronous Hopfield network . . . 882
21.1.2.2 Discrete asynchronous Hopfield network . . . 882
21.1.2.3 Analog Hopfield network . . . 882
21.1.2.4 Boltzmann machine . . . 883
21.1.2.5 Cauchy machine . . . 883
21.1.2.6 Hybrid Scheme . . . 884
21.1.3 Approximation with non-linear regression . . . 885
21.1.3.1 The method . . . 885
21.1.3.2 Defining the network . . . 885
21.1.3.3 The Lagrangian method . . . 886
21.2 Solving combinatorial optimisation problems . . . 887
21.2.1 Standard methods . . . 888
21.2.1.1 Formalising the problem . . . 888
21.2.1.2 Some solutions . . . 889
21.2.2 Solving the TSP with ML . . . 891
21.2.2.1 A review . . . 891
21.2.2.2 The problem . . . 892
21.2.3 The REINFORCE algorithm . . . 892
21.2.4 Applying reinforcement learning . . . 893

VI Applications **896**

22 Mathematical Finance **897**
22.1 Option pricing . . . 897
22.1.1 The framework . . . 897
22.1.1.1 Some definitions . . . 897
22.1.1.1.1 Asset returns . . . 897
22.1.1.1.2 Derivatives . . . 898
22.1.1.2 Defining the replicating portfolio . . . 898
22.1.1.3 The BS-formula . . . 901
22.1.1.4 The implied volatility . . . 902
22.1.1.5 The no-arbitrage conditions . . . 904
22.1.1.6 Characterising the smile dynamics . . . 905
22.1.1.6.1 The regimes of volatility . . . 905
22.1.1.6.2 Space homogeneity versus space inhomogeneity . . . 906
22.1.1.7 Pricing formulation . . . 906
22.1.1.8 The intrinsic risk . . . 907
22.1.1.8.1 Problems with continuous models . . . 907
22.1.1.8.2 Accounting for risk . . . 907
22.1.1.8.3 Example: the PnL explain . . . 908
22.1.1.9 Replicating strategy . . . 908

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

22.1.2 The calibration problem . . . 910
22.1.2.1 The general idea . . . 910
22.1.2.1.1 An ill-posed inverse problem . . . 910
22.1.2.1.2 Description . . . 910
22.1.2.1.3 Measures of pricing errors . . . 910
22.1.2.2 The problem . . . 911
22.1.2.2.1 Moments estimation . . . 911
22.1.2.2.2 The choice of a volatility model . . . 911
22.1.2.2.3 Accounting for no-arbitrage . . . 912
22.1.2.3 The standard approach . . . 912
22.1.2.3.1 Using prices . . . 912
22.1.2.3.2 From prices to volatilities . . . 912
22.1.2.3.3 A global optimum . . . 914
22.1.3 Trading strategies . . . 914
22.2 Supervised learning . . . 914
22.2.1 Fitting vanilla prices . . . 915
22.2.1.1 Fitting model prices . . . 915
22.2.1.1.1 The tests . . . 916
22.2.1.2 Fitting market prices . . . 916
22.2.1.2.1 The tests . . . 917
22.2.1.2.2 Results . . . 917
22.2.1.2.3 Analysis and solutions . . . 917
22.2.1.3 Completing market prices . . . 918
22.2.1.3.1 The problem . . . 918
22.2.1.3.2 Generating arbitrage-free data . . . 918
22.2.2 Calibration . . . 919
22.2.2.1 Description . . . 919
22.2.2.1.1 An overview . . . 919
22.2.2.1.2 Relationship between implied volatilities and option prices . . . 919
22.2.2.1.3 Pointwise learning . . . 920
22.2.2.1.4 Image-based implicit learning . . . 920
22.2.2.2 Problems . . . 921
22.2.3 Fitting exotic prices . . . 922
22.2.3.1 Description . . . 922
22.2.3.2 Problems . . . 922
22.2.3.2.1 Data . . . 922
22.2.3.2.2 Labelling the data . . . 923
22.2.3.2.3 Defining the range for the explanatory variables . . . 923
22.2.3.2.4 Measuring performance . . . 924
22.2.3.2.5 Additional parameters . . . 924
22.2.3.2.6 One network per payoff . . . 924
22.2.3.3 Risk analysis . . . 924
22.2.3.3.1 Events . . . 924
22.2.3.3.2 Extreme market behaviours . . . 925
22.3 Reinforcement learning . . . 925
22.3.1 Pricing options . . . 926
22.3.1.1 Formulation . . . 926
22.3.1.2 The algorithm . . . 927
22.3.2 Hedging options . . . 928
22.3.2.1 Deterministic BS-model . . . 928

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

22.3.2.1.1 Pricing in discrete time . . . 928
22.3.2.1.2 The value function . . . 928
22.3.2.1.3 The Q-value function . . . 931
22.3.2.2 Learning the optimal policy . . . 932
22.3.2.2.1 Known model . . . 932
22.3.2.2.2 Unknown model . . . 933
22.3.2.3 Non-normal rate model . . . 934
22.3.2.3.1 The assumption of normal returns . . . 935
22.3.2.3.2 Beyond normal returns . . . 935
22.3.2.3.3 The modified risk-averse price . . . 935
22.3.2.3.4 The modified value function . . . 936
22.4 Appendices . . . 938
22.5 Local risk minimisation . . . 938
22.5.1 Portfolio evaluation . . . 938
22.5.1.1 Description . . . 938
22.5.1.2 Problems and solutions . . . 939
22.5.1.2.1 An overview . . . 939
22.5.1.2.2 The recursive portfolio equation . . . 940
22.5.2 Optimal hedging . . . 941
22.5.2.1 Description . . . 941
22.5.2.2 Problems and solutions . . . 942
22.5.3 The risk-averse price . . . 942
22.5.4 The BS limit . . . 943
22.6 Miscellaneous . . . 944
22.6.1 Some facts on the BS-framework . . . 944
22.6.1.1 Defining the dynamics of European option prices . . . 944
22.6.1.2 The Greeks and their interpretation . . . 946
22.6.2 Some expansions . . . 948
22.6.2.1 Taylor series expansion . . . 948
22.6.2.1.1 Ordinary calculus . . . 948
22.6.2.1.2 Informal derivation . . . 948
22.6.2.2 Ito's formula . . . 949
22.6.3 The PAA methodology . . . 949
22.6.3.1 Definition . . . 949
22.6.3.2 PAA limitations . . . 950
22.6.3.3 Example . . . 951
22.6.3.4 Market practices . . . 951
22.6.4 Point on Kolmogorov-Compatibility . . . 952
22.7 The parametric MixVol model . . . 953
22.7.1 Description of the model . . . 953
22.7.2 Computing the Greeks . . . 955
22.7.3 Scenarios analysis . . . 955

VII Annexes **957**

23 Some mathematical facts **958**
23.1 Notation . . . 958
23.2 Miscellaneous . . . 958
23.2.1 Some functions . . . 959

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

23.2.1.1 Some optimums . . . 959
23.2.1.1.1 The maxima and minima . . . 959
23.2.1.1.2 The infimum and supremum . . . 959
23.2.1.1.3 Argument of the maximum . . . 960
23.2.1.2 The derivative . . . 960
23.2.1.3 The subgradient . . . 961
23.2.1.4 The kernel functions . . . 962
23.2.1.5 The Dirac function . . . 964
23.2.1.6 The multiset . . . 965
23.2.1.7 The sequence . . . 966
23.2.1.7.1 Convergence . . . 967
23.2.1.7.2 The series . . . 968
23.2.1.8 Fixed point . . . 968
23.2.2 Some facts on convex and concave analysis . . . 969
23.2.2.1 Convex functions . . . 969
23.2.2.2 Concave functions . . . 970
23.2.2.3 Some approximations . . . 971
23.2.3 Some notes on convolution . . . 971
23.2.3.1 Description . . . 971
23.2.3.2 Some properties . . . 973
23.2.3.3 Signal analysis . . . 974
23.2.3.3.1 Discrete convolution . . . 975
23.2.3.3.2 Correlation . . . 975
23.2.3.4 Multidimensional discrete convolution . . . 976
23.2.3.4.1 Description . . . 976
23.2.3.4.2 Separable signals . . . 976
23.2.3.4.3 Gaussian convolution . . . 977
23.3 Introduction to wavelet theory . . . 978
23.3.1 Some definitions . . . 978
23.3.1.1 Continuous Wavelet . . . 979
23.3.1.2 Discrete Wavelet . . . 979
23.3.2 Decomposition schemes . . . 980
23.3.2.1 Discrete Wavelet Transform . . . 980
23.3.2.2 Stationary Wavelet Transform . . . 981
23.3.2.3 Wavelet Packet Transform . . . 982
23.3.3 Denoising techniques . . . 983
23.4 Dynamical systems theory . . . 985
23.4.1 Introduction to dynamical systems . . . 985
23.4.1.1 An overview . . . 985
23.4.1.2 Description . . . 986
23.4.1.2.1 Linear systems . . . 987
23.4.1.2.2 Nonlinear systems . . . 987
23.4.2 Ergodic theory . . . 988
23.4.3 State space models . . . 989
23.4.3.1 An overview . . . 989
23.4.3.2 Linear models . . . 990
23.4.3.3 Non-linear models . . . 991

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

24 Some notions of probabilities **992**
24.1 Introduction to random variables . . . 992
24.1.1 Events as sets and their probabilities . . . 992
24.1.1.1 Sets of events . . . 992
24.1.1.2 Probability measures . . . 992
24.1.1.3 Conditional events . . . 993
24.1.1.4 Chain rule . . . 994
24.1.2 Defining random variables . . . 994
24.1.2.0.1 The joint probability distribution . . . 994
24.1.2.1 Discrete random variables . . . 995
24.1.2.2 Continuous random variables . . . 998
24.1.3 The characteristic function, moments and cumulants . . . 1000
24.1.3.1 Definitions . . . 1000
24.1.3.2 The first two moments . . . 1001
24.1.4 Conditional moments . . . 1002
24.1.4.1 Conditional expectation . . . 1003
24.1.4.2 Conditional variance . . . 1005
24.1.5 Multiple random variables . . . 1006
24.1.5.1 Discrete variables . . . 1007
24.1.5.2 Continuous variables . . . 1008
24.1.6 Some limits . . . 1009
24.2 Sampling data . . . 1010
24.2.1 Probability sampling . . . 1011
24.2.1.1 Simple random sampling . . . 1011
24.2.1.2 Systematic sampling . . . 1012
24.2.1.3 Stratified sampling . . . 1013
24.2.1.4 Cluster sampling . . . 1015
24.2.2 Non-probability sampling . . . 1016
24.2.2.1 Convenience sampling . . . 1016
24.2.2.2 Quota sampling . . . 1017
24.2.2.3 Judgement sampling . . . 1017
24.2.2.4 Minimax sampling . . . 1017
24.2.2.5 Accidental sampling . . . 1017
24.2.2.6 Voluntary sampling . . . 1018
24.2.2.7 Panel sampling . . . 1018
24.3 Some random sampling . . . 1018
24.3.1 The sample moments . . . 1018
24.3.2 Estimation of a ratio . . . 1021
24.3.3 Stratified random sampling . . . 1022
24.3.4 Geometric mean . . . 1026
24.4 Some continuous variables and their distributions . . . 1026
24.4.1 Some popular distributions . . . 1026
24.4.1.1 Uniform distribution . . . 1026
24.4.1.2 Exponential distribution . . . 1027
24.4.1.3 Gamma distribution . . . 1027
24.4.2 The Gaussian distribution . . . 1028
24.4.2.1 Linear . . . 1028
24.4.3 The exponential law and the Poisson law . . . 1029
24.4.4 The stable distributions . . . 1031
24.4.5 Power-law distributions . . . 1034

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

24.4.5.1 Definitions . . . 1034
24.5 Some discrete variables and their distributions . . . 1034
24.5.1 Some popular distributions . . . 1035
24.5.1.1 Multinomial distribution . . . 1035
24.6 Statistical models . . . 1035
24.6.1 Bayesian statistics . . . 1035
24.6.1.1 Bayesian rules . . . 1035
24.6.1.2 Bayesian analysis . . . 1036
24.6.2 Simple random walk . . . 1037
24.6.3 Some Markov properties . . . 1037
24.6.3.1 Some examples . . . 1039
24.6.3.1.1 Gibbs measure . . . 1039
24.6.3.1.2 The Ising model . . . 1040

25 Introduction to stochastic calculus 1041
25.1 An overview . . . 1041
25.1.1 Some definitions . . . 1041
25.1.1.1 stochastic processes . . . 1041
25.1.1.2 Local martingales and semi-martingales . . . 1042
25.1.1.3 The quadratic variation . . . 1043
25.1.1.3.1 The covariation of martingales . . . 1043
25.1.1.3.2 Some examples . . . 1044
25.1.2 Basic concepts . . . 1045
25.2 Introduction to Markov processes . . . 1054
25.2.1 Defining Markov processes . . . 1054
25.2.1.0.1 Strong Markov processes . . . 1056
25.2.2 Higher-dimensional Markov chains . . . 1057
25.2.2.1 2D Markov chains . . . 1058
25.2.2.2 Categorical data . . . 1060
25.2.2.2.1 First-order Markov chain . . . 1060
25.2.2.2.2 Higher-order Markov chain . . . 1062

26 Monte Carlo 1064
26.1 The dynamics of the state of the system . . . 1064
26.1.1 The stochastic differential equation . . . 1064
26.1.2 From Stratonovich equation to Ito equation . . . 1065
26.1.3 Discretisation of the SDE . . . 1066
26.1.3.1 The Euler scheem . . . 1067
26.1.3.2 Justification of the Euler scheme . . . 1067
26.1.4 Generating trajectories . . . 1067
26.1.4.1 Multivariate normals . . . 1067
26.1.4.2 One-dimension . . . 1068
26.1.4.3 Multi-dimensions . . . 1068
26.1.5 Construction of the Brownian bridge . . . 1069
26.1.5.1 Construction . . . 1069
26.1.5.2 Justification . . . 1069
26.1.6 Simulating the state process . . . 1070
26.2 The Monte Carlo engine . . . 1071
26.2.1 The setup of the Monte Carlo engine . . . 1071
26.2.2 Variance reduction techniques . . . 1072

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

26.2.2.1 Control variates . . . 1073
26.2.2.2 Perfect control variates . . . 1073
26.2.2.3 Using the hedge as control variates . . . 1074
26.2.2.4 Importance sampling . . . 1075
26.2.2.5 Antithetic variables . . . 1076
26.2.3 The speed of convergence . . . 1076
26.2.3.1 The Richardson extrapolation . . . 1076
26.2.3.2 The Romberg methods for Euler and Milstein scheme . . . 1077
26.2.4 Weak error for path-dependent functionals . . . 1078
26.2.5 Romberg extrapolation for path-dependent functionals . . . 1079
26.2.5.1 The stepwise constant Euler scheme . . . 1079
26.2.5.2 The continuous Euler scheme . . . 1080

27 Multifractal analysis 1081
27.1 Introducing fractal analysis . . . 1081
27.1.1 A brief history . . . 1081
27.1.2 Presenting the results . . . 1083
27.2 Defining random fractals . . . 1086
27.2.1 The fractional Brownian motion . . . 1086
27.2.2 The multidimensional fBm . . . 1087
27.2.3 The fractional Gaussian noise . . . 1088
27.2.4 The fractal process and its distribution . . . 1088
27.3 Defining multifractality . . . 1089
27.3.1 Fractal measures and their singularities . . . 1090
27.3.2 Scaling analysis . . . 1093
27.3.3 Multifractal analysis . . . 1095
27.3.4 The wavelet transform and the thermodynamical formalism . . . 1097

28 Portfolio Theory 1099
28.1 Miscellaneous . . . 1099
28.1.1 The capital asset pricing model . . . 1099
28.1.1.1 Mean-variance criterion . . . 1099
28.1.1.1.1 Normal returns . . . 1099
28.1.1.1.2 Non-normal returns . . . 1100
28.1.1.2 Markowitz solution to the portfolio allocation problem . . . 1100
28.1.1.3 The expected growth rate . . . 1101
28.1.2 Risk and return analysis . . . 1102
28.1.2.1 Performance measures . . . 1102
28.1.2.1.1 The Sharpe ratio . . . 1103
28.1.2.2 Incorporating tail risk . . . 1103
28.1.2.3 Considering the value at risk . . . 1104
28.1.2.3.1 Introducing the value at risk . . . 1104
28.1.2.3.2 The reward to VaR . . . 1105
28.1.2.3.3 The conditional Sharpe ratio . . . 1105
28.2 Dynamical models . . . 1106
28.2.1 The discrete model . . . 1106
28.2.1.1 Description . . . 1106
28.2.1.2 Example: Two assets model . . . 1108
28.2.1.3 Minimal risk in incomplete market . . . 1109
28.2.2 The continuous model . . . 1111

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

28.2.2.1 The stochastic integral . . . 1111
28.2.2.2 Description . . . 1112
28.2.2.3 The self-financing condition . . . 1113
28.2.2.3.1 Description . . . 1113
28.2.2.3.2 Discounted prices . . . 1114
28.2.2.4 Valuation and dynamics . . . 1114
28.2.2.5 Some examples . . . 1115
28.2.3 The BS-formula . . . 1115
28.2.3.1 The historical measure . . . 1115
28.2.3.2 The risk-neutral measure . . . 1118
28.2.3.3 The dynamics of the price . . . 1119
28.2.4 The replicating portfolio . . . 1120
28.2.4.1 Complete market . . . 1121
28.2.4.2 Incomplete market . . . 1122
28.3 Portfolio optimisation . . . 1124
28.3.1 Dynamic programming . . . 1124
28.3.1.1 Defining the problem . . . 1124
28.3.1.2 Solving the HJB equation . . . 1125
28.3.2 The martingale approach . . . 1127
28.3.2.1 Admissible strategies . . . 1127
28.3.2.2 Existence of an optimal pair . . . 1128
28.3.3 Dynamic programming: Some proofs . . . 1130
28.3.3.1 The principle of dynamic programming . . . 1130
28.3.3.2 The HJB equation . . . 1130
28.3.3.3 Sufficient conditions for an optimal solution . . . 1131
28.4 Introduction to the BSDE . . . 1131
28.4.1 Description . . . 1132
28.4.1.1 The linear BSDE . . . 1133
28.4.1.2 The flow property . . . 1134
28.4.2 Control problems . . . 1134
28.4.2.1 Value function expressed as BSDE . . . 1135
28.4.2.2 BSDE expressed as value function . . . 1136
28.4.2.3 Variational formulation of recursive utility . . . 1137
28.4.3 Superstrategy . . . 1137

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Part I

Knowledge representation

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 1

Presentation

1.1 Models

Models describe our beliefs about how the world functions; they explain a system and study the effects of different components to make predictions about behaviour.
See books by Gershenfeld [1998], Rutherford [2003], Bender [2003], Dubois [2018], among others.

1.1.1 Mathematical models

1.1.1.1 Description

A mathematical model is a description of a system using mathematical concepts and language, and mathematical modelling is the process of developing a mathematical model. In general, mathematical models are composed of relationships and variables. That is, a model describes a system by a set of variables and a set of equations that establish relationships between the variables. Variables may be of many types such as real or integer numbers, boolean values or strings. While the variables represent some properties of the system, the actual model is the set of functions that describe the relations between the different variables. These relationships can be described by operators, such as algebraic operators, functions, differential operators, etc. When studying models, it is helpful to identify broad categories of models. Classification of individual models into these categories tells us immediately some of the essentials of their structure. Several classification criteria can be used for mathematical models according to their structure. For instance, static and dynamic model, or deterministic and stochastic model. Static models do not explicitly account for time-variation, whereas dynamic models explicitly deal with time-variation interaction. In a deterministic model, all mathematical and logical relationship between the elements are fixed. As a result, these relationships completely determine the solutions. In a stochastic model, at least one variable is random (see Rubinstein [1981]).

In business and engineering, mathematical models may be used to maximise a certain output. For example, the system under consideration may require certain inputs, and the relation between inputs and outputs may depend on other variables such as decision variables, state variables, exogenous variables, and random variables. Decision variables are sometimes known as independent variables. Exogenous variables are sometimes known as parameters or constants. The variables are not independent of each other as the state variables are dependent on the decision, input, random, and exogenous variables. Furthermore, the output variables are dependent on the state of the system (represented by the state variables).
As opposed to observable variables, latent variables are variables that are not directly observed but are rather inferred (through a mathematical model) from other variables that are observed (directly measured). Mathematical models aiming to explain observed variables in terms of latent variables are called latent variable models (LVMs). Latent variable models are used in many disciplines, including psychology, demography, economics, engineering, medicine,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

physics, machine learning/artificial intelligence, bioinformatics, natural language processing, econometrics, management and the social sciences.
Sometimes, latent variables correspond to aspects of physical reality, which could in principle be measured, but may not be for practical reasons. In this situation, the term hidden variables is commonly used (reflecting the fact that the variables are really there, but hidden). Other times, latent variables correspond to abstract concepts, like categories, behavioural or mental states, or data structures. The terms hypothetical variables or hypothetical constructs may be used in these situations.
One advantage of using latent variables is that they can serve to reduce the dimensionality of data. A large number of observable variables can be aggregated in a model to represent an underlying concept, making it easier to understand the data. In this sense, they serve a function similar to that of scientific theories. At the same time, latent variables link observable (sub-symbolic) data in the real world to symbolic data in the modelled world.
Some common methods for inferring latent variables are as follows:

- Hidden Markov models
- Factor analysis
- Principal component analysis (PCA)
- Partial least squares regression
- Latent semantic analysis and probabilistic latent semantic analysis
- EM algorithms

Bayesian statistics is often used for inferring latent variables. Examples are:

- Latent Dirichlet allocation
- The Chinese restaurant process is often used to provide a prior distribution over assignments of objects to latent categories.
- The Indian buffet process is often used to provide a prior distribution over assignments of latent binary features to objects.

Objectives and constraints of the system and its users can be represented as functions of the output variables or state variables. The objective functions will depend on the perspective of the model's user. Depending on the context, an objective function is also known as an index of performance, as it is some measure of interest to the user. Although there is no limit to the number of objective functions and constraints a model can have, using or optimising the model becomes more involved (computationally) as the number increases.

Mathematical modelling problems are often classified into black box or white box models, according to how much a priori information on the system is available. A black-box model is a system of which there is no a priori information available, while a white-box model (also called glass box or clear box) is a system where all necessary information is available. Practically all systems are somewhere between the black-box and white-box models, so that this concept is only useful as an intuitive guide for deciding which approach to take. It is preferable to use as much a priori information as possible to make the model more accurate. If you use this information correctly, then the model will behave correctly. Often the a priori information comes in forms of knowing the type of functions relating different variables. In black-box models one tries to estimate both the functional form of relations between variables and the numerical parameters in those functions. In that setting, we could try to use functions as general as possible to cover all different models. Examples of such general functions are neural networks (NNs) which usually do not make assumptions about incoming data, and the NARMAX (Nonlinear AutoRegressive Moving Average model with eXogenous inputs)

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

algorithms which were developed as part of nonlinear system identification. The latter can be used to select the model terms, determine the model structure, and estimate the unknown parameters in the presence of correlated and nonlinear noise. The advantage of NARMAX models compared to neural networks is that it produces models that can be written down and related to the underlying process, whereas neural networks produce an approximation that is opaque (interpretability issues).

One way around, is to incorporate subjective information into a mathematical model based on intuition, experience, or expert opinion, or based on convenience of mathematical form. For instance, Bayesian statistics provides a theoretical framework for incorporating such subjectivity into a rigorous analysis: we specify a prior probability distribution (which can be subjective), and then update this distribution based on empirical data. However, while adding complexity usually improves the realism of a model, it can make the model difficult to understand and analyse, and can also pose computational problems, including numerical instability. In general, model complexity involves a trade-off between simplicity and accuracy of the model. It leads to Occam's razor principle which states that among models with roughly equal predictive power, the simplest one is the most desirable.

Any model which is not pure white-box contains some parameters that can be used to fit the model to the system it is intended to describe. If the modelling is done by an artificial neural network or other machine learning, the optimisation of parameters is called training, while the optimisation of model hyperparameters is called tuning and often uses cross-validation. In more conventional modelling through explicitly given mathematical functions, parameters are often determined by curve fitting.

1.1.1.2 Statistical model

A statistical model is a mathematical model that embodies a set of statistical assumptions concerning the generation of sample data (and similar data from a larger population). A statistical model represents, often in considerably idealised form, the data-generating process. A statistical model is usually specified as a mathematical relationship between one or more random variables and other non-random variables. As such, a statistical model is a formal representation of a theory.
A statistical model is usually thought of as a pair $(S, \mathcal{P})$, where S is the set of possible observations (sample space), and $\mathcal{P}$ is a set of probability distributions on S (see McCullagh [2002]). It is assumed that there is a "true" probability distribution induced by the process that generates the observed data. We choose $\mathcal{P}$ to represent a set (of distributions) which contains a distribution that adequately approximates the true distribution. Note that we do not require that $\mathcal{P}$ contains the true distribution. The set $\mathcal{P}$ is almost always parameterised: $\mathcal{P} = \{P_\theta : \theta \in \Theta\}$. The set Θ defines the parameters of the model. A parameterisation is generally required to have distinct parameter values give rise to distinct distributions, $P_{\theta_1} = P_{\theta_2} \Rightarrow \theta_1 = \theta_2$ must hold (in other words, it must be injective). A parameterisation that meets the requirement is said to be identifiable.
Given a statistical model $(S, \mathcal{P})$ with $\mathcal{P} = \{P_\theta : \theta \in \Theta\}$, it is said to be parametric if Θ has a finite dimension. In notation, we write that $\Theta \subseteq \mathbb{R}^k$, where k is a positive integer called the dimension of the model. A statistical model is nonparametric if the parameter set Θ is infinite dimensional. A statistical model is semiparametric if it has both finite-dimensional and infinite-dimensional parameters. Formally, if k is the dimension of Θ and n is the number of samples, both semiparametric and nonparametric models have $k \to \infty$ as $n \to \infty$. If $\frac{k}{n} \to 0$ as $n \to \infty$, then the model is semiparametric, otherwise, the model is nonparametric. Parametric models are by far the most commonly used statistical models.

1.1.1.3 Defining a theory

In mathematical logic, a theory is a set of sentences in a formal language (see Curry [2010]). The construction of a theory begins by specifying a definite non-empty conceptual class $\mathcal{E}$ where the elements are called statements. These initial statements are often called the primitive elements or elementary statements of the theory, to distinguish them from other statements which may be derived from them. A theory $\mathcal{T}$ is a conceptual class consisting of certain of

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

these elementary statements. The elementary statements which belong to $\mathcal{T}$ are called the elementary theorems of $\mathcal{T}$ and said to be true. Thus, a theory is a way of designating a subset of $\mathcal{E}$ which consists entirely of true statements. It stipulates that the truth of any of its elementary statements is not known without reference to $\mathcal{T}$, so that the same elementary statement may be true with respect to one theory, and not true with respect to another.
A theory $\mathcal{S}$ is a sub-theory of a theory $\mathcal{T}$ if $\mathcal{S}$ is a subset of $\mathcal{T}$. On the other hand, if $\mathcal{T}$ is a subset of $\mathcal{S}$ then $\mathcal{S}$ is an extension or super-theory of $\mathcal{T}$. A theory is said to be a deductive theory if $\mathcal{T}$ is an inductive class. That is, its content is based on some formal deductive system and some of its elementary statements are taken as axioms. In a deductive theory, any sentence which is a logical consequence of one or more of the axioms is also a sentence of that theory.
A syntactically consistent theory is a theory from which not every sentence in the underlying language can be proven (with respect to some deductive system which is usually clear from context). In a deductive system (such as first-order logic) that satisfies the principle of explosion, this is equivalent to requiring that there is no sentence φ such that both φ and its negation can be proven from the theory. A satisfiable theory is a theory that has a model. This means that there is a structure M satisfying every sentence in the theory. Any satisfiable theory is syntactically consistent, because the structure satisfying the theory will satisfy exactly one of φ and the negation of φ for each sentence φ. A consistent theory is sometimes defined to be a syntactically consistent theory, and sometimes defined to be a satisfiable theory. In the case of first-order logic, it follows from the completeness theorem that the two meanings coincide. In other logics, such as second-order logic, there are syntactically consistent theories that are not satisfiable, such as ω-inconsistent theories. A complete theory is a consistent theory $\mathcal{T}$ such that for every sentence φ in its language, either φ is provable from $\mathcal{T}$ or $\mathcal{T} \cup \{\varphi\}$ is inconsistent. For theories closed under logical consequence, this means that for every sentence φ, either φ or its negation is contained in the theory. An incomplete theory is a consistent theory that is not complete.
Each structure has several associated theories. The complete theory of a structure A is the set of all first-order sentences over the signature of A which are satisfied by A. It is denoted by $Th(A)$. More generally, the theory of K a class of σ-structures, is the set of all first-order σ-sentences that are satisfied by all structures in K, and is denoted by $Th(K)$. Clearly $Th(A) = Th(\{A\})$. These notions can also be defined with respect to other logics. For each σ-structure A, there are several associated theories in a larger signature $\sigma^{'}$ that extends σ by adding one new constant symbol for each element of the domain of A. The cardinality of $\sigma^{'}$ is thus the larger of the cardinality of σ and the cardinality of A.

1.1.1.4 Mathematical representation

Duval [2006] stated that mathematical objects (ideas, concepts and relations) can only be accessed through representation and its activities are affected by how the representation is used. Formally, a mathematical representation is a very general relationship that expresses similarities (or equivalences) between mathematical objects or structures. For instance, a collection Y of mathematical objects may be said to represent another collection X of objects, provided that the properties and relationships existing among the representing objects y_i conform, in some consistent way, to those existing among the corresponding represented objects x_i. Specifically, given a set π of properties and relations, a π-representation of some structure X is a structure Y that is the image of X under a homomorphism that preserves π. The label representation is sometimes also applied to the homomorphism itself (such as group homomorphism in group theory).
Mathematical entities can be represented in multiple ways (see Samsuddin [2018]):
Representation theory is a branch of mathematics that studies abstract algebraic structures by representing their elements as linear transformations of vector spaces and studies modules over these abstract algebraic structures. a representation makes an abstract algebraic object more concrete by describing its elements by matrices and their algebraic operations (for example, matrix addition, matrix multiplication). The theory of matrices and linear operators is well-understood, so representations of more abstract objects in terms of familiar linear algebra objects helps glean properties and sometimes simplify calculations on more abstract theories.
There are many other uses of the term representation throughout mathematics. Under certain circumstances, a single function $f : X \to Y$ is at once an isomorphism from several mathematical structures on X. Since each of those structures may be thought of as a meaning of the image Y this phenomenon is called polysemy. In linguistics, polysemy is the capacity for a word or phrase to have multiple meanings, usually related by contiguity of meaning within

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

a semantic field.
When it comes to knowledge, a knowledge representation (KR) is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity [1] to determine consequences by thinking rather than acting, that is, by reasoning about the world rather than taking action in it.

1.1.2 Algorithms

1.1.2.1 An overview

The mathematics of George Boole (1847, 1854), Gottlob Frege (1879), and Giuseppe Peano (1888-1889) reduced arithmetic to a sequence of symbols manipulated by rules (see Section (2.1)). Frege proposed a "formula language", that is a language written with special symbols constructed from specific symbols that are manipulated according to definite rules. The work of Frege was further simplified and amplified by Alfred North Whitehead and Bertrand Russell in their Principia Mathematica (PM) (1910-1913) (see Irvine [2003]). Turing [1936-37] and Post [1936] (independently) described a process of men-as-computers working on computations model. They started with an analysis of a human computer that they whittles down to a simple set of basic motions and states of mind. But Turing continued a step further and created a machine as a model of computation of numbers. A few years later, Turing expanded his analysis and described a function to be "effectively calculable" if its values can be found by some purely mechanical process (calculable by a machine).

In mathematics and computer science, an algorithm is an unambiguous specification of how to solve a class of problems. Kleene [1943] proposed the following definition (Thesis I known as the Church-Turing thesis):

Definition 1.1.1 *An algorithm describes a procedure, performable for each set of values of the independent variables, which procedure necessarily terminates and in such manner that from the outcome we can read a definite answer, "yes" or "no", to the question "is the predicate value true?".*

According to Rogers [1967], algorithms can perform calculation, data processing and automated reasoning tasks. An informal definition proposed by Stone [1972] is as follows:

Definition 1.1.2 *An algorithm is a set of rules that precisely defines a sequence of operations.*

As an effective method, an algorithm can be expressed within a finite amount of space and time and in a well-defined formal language for calculating a function. Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states, eventually producing output and terminating at a final ending state. An algorithm has one or more outputs, that is, quantities which have a specified relation to the inputs (see Knuth [1973]). The transition from one state to the next is not necessarily deterministic; some algorithms, known as randomised algorithms, incorporate random input.
The concept of algorithm has existed for centuries. Babylonian clay tablets describe and employ algorithmic procedures to compute the time and place of significant astronomical events. Greek mathematicians used algorithms in, for example, the sieve of Eratosthenes for finding prime numbers and the Euclidean algorithm for finding the greatest common divisor of two numbers.
A partial formalisation of what would become the modern concept of algorithm began with attempts at solving the Entscheidungsproblem (decision problem) posed by David Hilbert in 1928. Later formalisations were framed as attempts to define effective calculability or effective method. Those formalisations included the Godel-Herbrand-Kleene recursive functions of 1930, 1934 and 1935, Alonzo Church's lambda calculus of 1936, Emil Post's Formulation 1 of 1936 (see Post [1936]), and Alan Turing's Turing machines (see Turing [1936-37] [1939]).

[1] An entity is something that exists apart from other things, having its own independent existence.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.1.2.2 Description

An algorithm is a set of rules that precisely defines a sequence of operations. For instance, most computer programs contain algorithms that detail the specific instructions a computer should perform (in a specific order) to carry out a specified task. Thus, an algorithm can be considered to be any sequence of operations that can be simulated by a Turing-complete system (see Minsky [1967], Savage (1987) and Gurevich [2000]).
In computer systems, an algorithm is an instance of logic written in software by software developers, to be effective for the intended target computer(s) to produce output from given (perhaps null) input. An optimal algorithm, even running in old hardware, would produce faster results than a non-optimal (higher time complexity) algorithm for the same purpose, running in more efficient hardware; that is why algorithms, like computer hardware, are considered technology.
Assuming the premises of imperative programming, when an algorithm is associated with processing information, data can be read from an input source, written to an output device and stored for further processing. Stored data are regarded as part of the internal state of the entity performing the algorithm. In practice, the state is stored in one or more data structures. Thus, the algorithm must be rigorously defined and the order of computation is crucial to the functioning of the algorithm. Instructions are usually assumed to be listed explicitly, and are described as starting from the top and going down to the bottom.

Algorithms can be expressed in many kinds of notation such as

- Natural languages: Natural language expressions of algorithms tend to be verbose and ambiguous, and are rarely used for complex or technical algorithms.
- Pseudocode
- Flowcharts
- Drakon-charts
- Programming languages: They are primarily intended for expressing algorithms in a form that can be executed by a computer but are often used as a way to define or document algorithms.
- Control tables (processed by interpreters): See more at finite-state machine, state transition table and control table.

Pseudocode, flowcharts, drakon-charts and control tables are structured ways of expressing algorithms that avoid many of the ambiguities common in natural language statements.

Algorithm design is a method or mathematical process for problem-solving and engineering algorithms. It is part of many solution theories of operation research, such as dynamic programming and divide-and-conquer. Techniques for designing and implementing algorithm designs are also called algorithm design patterns, such as the template method pattern and decorator pattern. One of the most important aspects of algorithm design is the creation of an algorithm that has an efficient run-time, also known as its Big O. Typical steps in the development of an algorithm are as follows:

- Problem definition
- Development of a model
- Specification of the algorithm
- Designing an algorithm
- Checking the correctness of the algorithm
- Analysis of algorithm

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Implementation of algorithm
- Program testing
- Documentation preparation

Even though algorithms are intended to be implemented as computer programs, they are also implemented by other means, such as in a biological neural network (for example, the human brain implementing arithmetic or an insect looking for food), in an electrical circuit, or in a mechanical device.

1.1.2.3 Classification

For a given function, multiple algorithms may exist. There are various ways of classifying algorithms, each with its own merits. Some of these ways are:

- Implementation: recursion; logical; serial, parallel or distributed; deterministic or non-deterministic, exact or approximate; quantum algorithm.
- Design paradigm: brute force or exhaustive search; divide and conquer; search and enumeration; randomised algorithm; reduction of complexity; back tracking.
- Optimisation problems
- Field of study: Some example classes are search algorithms, sorting algorithms, merge algorithms, numerical algorithms, graph algorithms, string algorithms, computational geometric algorithms, combinatorial algorithms, medical algorithms, machine learning, cryptography, data compression algorithms and parsing techniques.
- Complexity: Algorithms can be classified by the amount of time they need to complete compared to their input size:
  - Constant time: if the time needed by the algorithm is the same, regardless of the input size, such as an access to an array element.
  - Linear time: if the time is proportional to the input size.
  - Logarithmic time: if the time is a logarithmic function of the input size, such as binary search algorithm.
  - Polynomial time: if the time is a power of the input size. For example, the bubble sort algorithm has quadratic time complexity.
  - Exponential time: if the time is an exponential function of the input size, such as Brute-force search.

Note, in the case of optimisation problems there is a more specific classification of algorithms (see details in Section (18.1.1)). An algorithm for such problems may fall into one or more of the general categories described above as well as into one of the following:

- Linear programming: When searching for optimal solutions to a linear function, bound to linear equality and inequality constraints, the constraints of the problem can be used directly in producing the optimal solutions.
- Dynamic programming: When a problem shows optimal substructures, meaning the optimal solution to a problem can be constructed from optimal solutions to subproblems, and overlapping subproblems, meaning the same subproblems are used to solve many different problem instances, a quicker approach called dynamic programming avoids recomputing solutions that have already been computed. Dynamic programming and memoization (it is an optimisation technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again.) go together. The main difference between dynamic programming and divide and conquer is that subproblems are more or less independent in divide and conquer, whereas subproblems overlap in dynamic programming.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- The greedy method: A greedy algorithm is similar to a dynamic programming algorithm in that it works by examining substructures, in this case not of the problem but of a given solution. Such algorithms start with some solution, which may be given or have been constructed in some way, and improve it by making small modifications. For some problems they can find the optimal solution while for others they stop at local optima, that is, at solutions that cannot be improved by the algorithm but are not optimum. The most popular use of greedy algorithms is for finding the minimal spanning tree where finding the optimal solution is possible with this method.
- The heuristic method: In optimisation problems, heuristic algorithms can be used to find a solution close to the optimal solution in cases where finding the optimal solution is impractical. These algorithms work by getting closer and closer to the optimal solution as they progress. In principle, if run for an infinite amount of time, they will find the optimal solution. Their merit is that they can find a solution very close to the optimal solution in a relatively short time. Such algorithms include local search, tabu search, simulated annealing, and genetic algorithms.

1.1.3 Computing

Computing is any activity that uses computers to manage, process, and communicate information. It includes development of both hardware and software. Computing also has other meanings that are more specific, based on the context in which the term is used. The discipline of computing is the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application.

1.1.3.1 Description

A computer is a machine that manipulates data according to a set of instructions called a computer program. The latter has an executable form that the computer can use directly to execute the instructions. Computing is any activity that uses computers. It is intimately tied to the representation of numbers. Wynn-Williams [1931] proposed using digital electronics for computing. Shannon [1938] introduced the idea of using electronics for Boolean algebraic operations (see Section (2.1)).
The principle of the modern computer was proposed by Turing [1936-37] who proposed a simple device that he called Universal Computing machine and that is now known as a universal Turing machine. He proved that such a machine is capable of computing anything that is computable by executing instructions (program) stored on tape, allowing the machine to be programmable. The fundamental concept of Turing's design is the stored program, where all the instructions for computing are stored in memory. Von Neumann acknowledged that the central concept of the modern computer was due to this paper. Turing machines are to this day a central object of study in theory of computation. Except for the limitations imposed by their finite memory stores, modern computers are said to be Turing-complete, which is to say, they have algorithm execution capability equivalent to a universal Turing machine.

Computing includes developing hardware and software, and using computers to manage, process, and communicate information for various purposes. Major computing disciplines include computer engineering, software engineering, computer science, information systems, and information technology. Computer hardware includes the physical, tangible parts or components of a computer, such as the cabinet, central processing unit, monitor, keyboard, computer data storage, graphics card, sound card, speakers and motherboard. Computer software is a collection of computer programs and related data that provides the instructions for telling a computer what to do and how to do it. Software refers to one or more computer programs and data held in the storage of the computer for some purposes. It is a set of programs, procedures, algorithms and its documentation concerned with the operation of a data processing system. Program software performs the function of the program it implements, either by directly providing instructions to the computer hardware or by serving as input to another piece of software. The term was coined to contrast with the old term hardware (meaning physical devices). In contrast to hardware, software is intangible. Hardware is typically directed by the software to execute any command or instruction. Application software is a computer software designed to help the user to perform specific tasks. A combination of hardware and software forms a usable computing system.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

An operating system (OS) is a system software that manages computer hardware, software resources, and provides common services for computer programs. Time-sharing OS schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources. A single-tasking system can only run one program at a time, while a multi-tasking operating system allows more than one program to be running in concurrency. This is achieved by time-sharing, where the available processor time is divided between multiple processes. These processes are each interrupted repeatedly in time slices by a task-scheduling subsystem of the operating system. Multi-tasking may be characterized in preemptive and co-operative types. A distributed operating system manages a group of distinct computers and makes them appear to be a single computer. When computers in a group work in cooperation, they form a distributed system. A real-time operating system is an operating system that guarantees to process events or data by a specific moment in time. A real-time operating system may be single- or multi-tasking, but when multitasking, it uses specialized scheduling algorithms so that a deterministic nature of behaviour is achieved. An event-driven system switches between tasks based on their priorities or external events while time-sharing operating systems switch tasks based on clock interrupts.

1.1.3.2 Control system

Computers are used as control systems for a wide variety of industrial and consumer devices. A control system manages, commands, directs, or regulates the behaviour of other devices or systems using control loops. It can range from a single home heating controller using a thermostat controlling a domestic boiler to large industrial control systems which are used for controlling processes or machines. A state is a description of the status of a system that is waiting to execute a transition. A transition is a set of actions to be executed when a condition is fulfilled or when an event is received.
The control unit (or control system or central controller) manages the various components of the computer. It reads and interprets (decodes) the program instructions, transforming them into control signals that activate other parts of the computer. Control systems in advanced computers may change the order of execution of some instructions to improve performance.
The Arithmetic logic unit (ALU) is capable of performing two classes of operations: arithmetic and logic. The set of arithmetic operations that a particular ALU supports may be limited to addition and subtraction, or might include multiplication, division, trigonometry functions such as sine, cosine, and square roots. Some can only operate on whole numbers (integers) while others use floating point to represent real numbers, albeit with limited precision. An ALU may also compare numbers and return boolean truth values (true or false) depending on whether one is equal to, greater than or less than the other. Logic operations involve Boolean logic: AND, OR, XOR, and NOT. These can be useful for creating complicated conditional statements and processing boolean logic (see Section (2.1)).
The control unit, ALU, and registers are collectively known as a central processing unit (CPU). A key component common to all CPUs is the program counter, a special memory cell (a register) that keeps track of which location in memory the next instruction is to be read from. A simplified description of a control system is as follows:

1. Read the code for the next instruction from the cell indicated by the program counter.
2. Decode the numerical code for the instruction into a set of commands or signals for each of the other systems.
3. Increment the program counter so it points to the next instruction.
4. Read whatever data the instruction requires from cells in memory (or perhaps from an input device). The location of this required data is typically stored within the instruction code.
5. Provide the necessary data to an ALU or register.
6. If the instruction requires an ALU or specialized hardware to complete, instruct the hardware to perform the requested operation.
7. Write the result from the ALU back to a memory location or to a register or perhaps an output device.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

8. Jump back to step (1).

Since the program counter is (conceptually) just another set of memory cells, it can be changed by calculations done in the ALU. Instructions that modify the program counter are often known as jumps and allow for loops (instructions that are repeated by the computer) and often conditional instruction execution (both examples of control flow).

1.1.3.3 Computer cluster

The desire to get more computing power and better reliability by orchestrating a number of low-cost commercial off-the-shelf computers has given rise to a variety of architectures and configurations.

A computer cluster is a set of loosely or tightly connected computers working together. They can be viewed as a single system. Unlike grid computers, computer clusters have each node set to perform the same task, controlled and scheduled by software. They are usually deployed to improve performance and availability over that of a single computer, while typically being much more cost-effective than single computers of comparable speed or availability (see Pfister [1998]). The components of a cluster are usually connected to each other through fast local area networks, with each node (computer used as a server) running its own instance of an operating system. In most circumstances, all of the nodes use the same hardware and the same operating system. The activities of the computing nodes are orchestrated by clustering middleware, a software layer that sits atop the nodes and allows the users to treat the cluster as by and large one cohesive computing unit. For example, via a single system image concept.

A cluster manager is a backend graphical user interface (GUI), or, command-line software, that runs on one or all cluster nodes. It works together with a cluster management agent. These agents run on each node of the cluster to manage and configure services, a set of services, or to manage and configure the complete cluster server itself. In some cases the cluster manager is mostly used to dispatch work for the cluster (or cloud) to perform. In this last case a subset of the cluster manager can be a remote desktop application that is used not for configuration but just to send work and get back work results from a cluster. In other cases the cluster is more related to availability and load balancing than to computational or specific service clusters.

A distributed data store is a computer network where information is stored on more than one node, often in a replicated fashion. It is usually specifically used to refer to either a distributed database where users store information on a number of nodes, or a computer network in which users store information on a number of peer network nodes. Distributed databases are usually non-relational databases that enable a quick access to data over a large number of nodes. Examples of limited distributed databases are Google's Bigtable, which is much more than a distributed file system or a peer-to-peer network, Amazon's Dynamo and Microsoft Azure Storage.

1.1.3.4 Orchestration

In computer science, a container is a class, a data structure, or an abstract data type (ADT) whose instances are collections of other objects. They store objects in an organised way that follows specific access rules. The size of the container depends on the number of objects (elements) it contains. Underlying (inherited) implementations of various container types may vary in size and complexity, and provide flexibility in choosing the right implementation for any given scenario. Container are characterized by the following three properties:

1. Access: The way of accessing the objects of the container. In the case of arrays, access is done with the array index. In the case of stacks, access is done according to the LIFO (last in, first out) order and in the case of queues it is done according to the FIFO (first in, first out) order.
2. Storage: The way of storing the objects of the container.
3. Traversal: The way of traversing the objects of the container.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies. He can then ship it all out as one package.

Orchestration is the automated configuration, coordination, and management of computer systems and software (see Erl [2005]). It is often discussed in the context of service-oriented architecture, virtualisation, provisioning, converged infrastructure and dynamic datacenter topics. In that case, it is about aligning the business request with the applications, data, and infrastructure. The main difference between a workflow automation and an orchestration is that workflows are processed and completed as processes within a single domain for automation purposes, whereas orchestration includes a workflow and provides a directed action towards larger goals and objectives.

On ordinary operating systems (OS) for personal computers, a computer program can see (but not always access) all the system's resources. They include:

- Hardware capabilities that can be employed, such as the CPU and the network connection.
- Data that can be read or written, such as files, folders and network shares.
- Connected peripherals it can interact with, such as webcam, printer, scanner, or fax.

The operating system may be able to allow or deny access to such resources based on which program requests them and the user account in the context of which it runs. The operating system may also hide those resources, so that when the computer program enumerates them, they do not appear in the enumeration results. Nonetheless, the computer program has interacted with those resources and the operating system has managed an act of interaction.
With operating-system-virtualisation, or containerisation, it is possible to run programs within containers, to which only parts of these resources are allocated. A program expecting to see the whole computer, once run inside a container, can only see the allocated resources and believes them to be all that is available. Several containers can be created on each operating system, to each of which a subset of the computer's resources is allocated. Each container may contain any number of computer programs. These programs may run concurrently or separately, and may even interact with one another.

Large organisations must ensure availability, load balancing, scale up and down based on used load. As a result, they use multiple containers for every application. Container orchestration tool (or engine) automates deploying, scaling, load balancing, and managing containerised application or group of servers. Some examples are Kubernetes, Docker Swarm, Apache Mesos Marathon.

OS-level virtualisation is an operating system paradigm in which the kernel allows the existence of multiple isolated user space instances. Such instances, called containers (Solaris, Docker), Zones (Solaris), virtual private servers (OpenVZ), partitions, virtual environments (VEs), virtual kernel (DragonFly BSD), or jails (FreeBSD jail or chroot jail), may look like real computers from the point of view of programs running in them. A computer program running on an ordinary operating system can see all resources (connected devices, files and folders, network shares, CPU power, quantifiable hardware capabilities) of that computer. However, programs running inside of a container can only see the container's contents and devices assigned to the container.

Docker is a set of platform as a service (PaaS) products using OS-level virtualisation to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. Docker can package an application and its dependencies in a virtual container that can run on any Linux server. It helps provide flexibility and portability enabling the application to be run in various locations, whether on-premises, in a public cloud, or in a private cloud. All containers are run by a single operating-system kernel and are thus more lightweight than virtual machines. It consists of three components: Software, Objects, Registries.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.1.3.5 Research

Nowadays, DNA-based computing and quantum computing are areas of active research in both hardware and software (such as the development of quantum algorithms). Based on classical physics, conventional computers operate by manipulating a large number of tiny transistors working essentially as on-off switches. These switches change state between cycles of the clock of the computer. The states are described by a long sequence of bits. Assuming N transistors, the computer can be in 2^N possible states. Thus, computation consists in switching some transistors of the computer between their on and off states according to a prescribed program. In quantum computing, the two-state circuit element is replaced by a quantum element called a quantum bit, or qubit, which also has two basic states. The electron's internal angular momentum, or spin, only has two possible projections on any coordinate axis: $+\frac{1}{2}$ and $-\frac{1}{2}$ (see Dyakonov [2019]). Given an axis, the two basic quantum states of the spin are denoted by $\uparrow$ and $\downarrow$. The spin state of an electron is described by a quantum mechanical wave function involving two complex numbers α and β (amplitudes). These numbers have a certain magnitude, and their squared magnitudes must add up to 1. It corresponds to the probabilities for the spin of the electron to be in the states $\uparrow$ and $\downarrow$. Thus, a qubit can be in any of a continuum of possible states, defined by the values of the amplitudes α and β. It is argued that a qubit can exists simultaneously in both of its $\uparrow$ and $\downarrow$ states. Assuming N qubits, the state of the system is described by 2^N complex numbers such that their squared magnitudes add up to 1. Here the state of the quantum computer is described by the values of the 2^N amplitudes, which are continuous parameters. This is to oppose with the 2^N possible states of an N bits conventional computer.

1.1.3.6 Knowledge representation

Knowledge representation is a field of computing that focuses on designing computer representations that capture information about the world that can be used to solve complex problems. This is because conventional procedural code is not the best formalism to use to solve complex problems. Knowledge representation makes complex software easier to define and maintain than procedural code and can be used in expert systems. Knowledge representation goes hand in hand with automated reasoning because one of the main purposes of explicitly representing knowledge is to be able to reason about that knowledge, to make inferences, assert new knowledge. The ultimate knowledge representation formalism in terms of expressive power and compactness is First Order Logic (FOL). There is no more powerful formalism than that used by mathematicians to define general propositions about the world. However, FOL has two drawbacks as a knowledge representation formalism: ease of use and practicality of implementation. Thus, a subset of FOL can be both easier to use and more practical to implement. IF-THEN rules provide a subset of FOL but a very useful one that is also very intuitive. Davis et al. [1993] outlined five distinct roles to analyse a knowledge representation framework:

- A knowledge representation (KR) is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, that is, by reasoning about the world rather than taking action in it.
- It is a set of ontological commitments, that is, an answer to the question: In what terms should I think about the world?
- It is a fragmentary theory of intelligent reasoning, expressed in terms of three components: (i) the representation's fundamental conception of intelligent reasoning; (ii) the set of inferences the representation sanctions, and (iii) the set of inferences it recommends.
- It is a medium for pragmatically efficient computation, that is, the computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency is supplied by the guidance a representation provides for organising information so as to facilitate making the recommended inferences.
- It is a medium of human expression, that is, a language in which we say things about the world.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.2 Machine learning

1.2.1 An overview

1.2.1.1 Artificial intelligence

As discussed in Sections (1.1.2) (1.1.3), Turing was highly influential in the development of theoretical computer science, providing a formalisation of the concepts of algorithm and computation with the Turing machine, which can be considered a model of a general-purpose computer. He is widely considered to be the father of theoretical computer science and artificial intelligence (AI).
Turing [1948] identified three ways by which human-competitive machine intelligence might be achieved. One of them is the genetical or evolutionary search by which a combination of genes is looked for, the criterion being the survival value. During his stay at the Victoria University of Manchester, he continued his work on Computing Machinery and Intelligence (see Turing [1950]) and stated that we could not find a good child-machine at the first attempt and that it should be thought in an evolutionary process. He perceived that one possibly productive approach to machine intelligence would involve an evolutionary process in which a description of a computer program undergoes progressive modification (mutation) under the guidance of natural selection.
Turing also understood the need to evaluate objectively the behaviour exhibited by machines, to avoid human biases when assessing their intelligence. He addressed the problem of artificial intelligence and proposed an imitation game that became known as the Turing test for machine intelligence, an attempt to define a standard for a machine to be called "intelligent". The idea was that a computer could be said to "think" if a human interrogator could not tell it apart, through conversation, from a human being (we call it below human intelligence). Turing suggested that rather than building a program to simulate the adult mind, it would be better rather to produce a simpler one to simulate a child's mind and then to subject it to a course of education. A reversed form of the Turing test is widely used on the Internet: the CAPTCHA test is intended to determine whether the user is a human or a computer. Along with other discoveries in neurobiology, information theory and cybernetics, it led researchers to consider the possibility of building an electronic brain. The first work recognised as AI was that of McCullouch et al. [1943] formal design of Turing complete artificial neurons.

The field of artificial intelligence (AI) research was born at a workshop at Dartmouth College in 1956. Attendees Allen Newell (CMU), Herbert Simon (CMU), John McCarthy (MIT), Marvin Minsky (MIT) and Arthur Samuel (IBM) became the founders and leaders of AI research. The field was founded on the claim that human intelligence can be so precisely described that a machine can be made to simulate it. Thus, early researchers developed algorithms that imitated step-by-step reasoning that humans use when they solve puzzles or make logical deductions.
Nowadays, artificial intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and other animals. It is applied when a machine mimics cognitive functions that humans associate with other minds, such as learning and problem solving. A typical AI perceives its environment and takes actions that maximise its chance of successfully achieving its goals (see Russell et al. [2009]). The goals can be explicitly defined, or induced. As machines become increasingly capable, tasks considered to require intelligence are often removed from the definition of AI, a phenomenon known as the AI effect. For instance, Tesler's Theorem states that AI is whatever has not been done yet.

1.2.1.2 Towards machine learning

In the early days of artificial intelligence (AI) as an academic discipline, some researchers were interested in having machines learn from data. They attempted to approach the problem with various symbolic methods, as well as what were then termed neural networks (NNs), which were mostly perceptrons (see McCullouch et al. [1943] and Rosenblatt [1958]) and other models that were later found to be reinventions of the generalised linear models of statistics (see Sarle [1994]). Thus, machine learning (ML) is a field of artificial intelligence (AI) that uses statistical techniques to give computer systems the ability to learn (progressively improve performance on a specific task) from

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

data, without being explicitly programmed (see Arthur Samuel [1959]). However, by 1980, expert systems (a computer system that emulates the decision-making ability of a human expert) had come to dominate AI, and statistics was out of favour. Neural networks research had been abandoned by AI and computer science around the same time. Nonetheless, this line of research was continued outside the AI/CS field, as connectionism, by researchers from other disciplines including Hopfield, Rumelhart and Hinton. Their main success came in the mid-1980s with the reinvention of backpropagation (see Rumelhart et al. [1986c]). Machine learning, reorganised as a separate field, started to flourish in the 1990s. The field changed its goal from achieving artificial intelligence to tackling solvable problems of a practical nature. It shifted focus away from the symbolic approaches it had inherited from AI, and toward methods and models borrowed from statistics and probability theory. It also benefited from the increasing availability of digitised information, and the ability to distribute it via the Internet.

Samuel [1983] used the term "human-competitive" in connection with evaluating the result of an automated problem solving method. He stated: the aim is to get machines to exhibit behaviour, which if done by humans, would be assumed to involve the use of intelligence.
An automatically created solution to a problem is human-competitive if it satisfies one or more of the eight following criteria:

1. The result was patented as an invention in the past, is an improvement over a patented invention, or would qualify today as a patentable new invention.
2. The result is equal or better than a result that was accepted as a new scientific result at the time when it was published in a peer-reviewed scientific journal.
3. The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognised panel of scientific experts.
4. The result is publishable in its own right as a new scientific result, independent of the fact that the result was mechanically created.
5. The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.
6. The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered.
7. The result solves a problem of indisputable difficulty in its field.
8. The result holds its own or wind a regulated competition involving human contestants.

Modifying Alan Turing's proposal (see Turing [1950]), in which the question "Can machines think?" with the question "Can machines do what we (as thinking entities) can do?", Tom M. Mitchell [1997] proposed a formal definition of the algorithms studied in the machine learning field:

Definition 1.2.1 *Learning from data*
A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

Nowadays we are talking about the great decoupling (see Harari [2016]) where intelligence is decoupling from consciousness to create a new type of non-consciousness intelligence that can perform some tasks (playing chess, driving cars, diagnosing diseases or identifying people) far better than humans. While there are several ways leading to super-intelligence, only some of them pass through the straits of consciousness. For armies and corporations intelligence (performing tasks) is mandatory but consciousness is optional.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.2.2 Objectives

1.2.2.1 Life is a sub-optimal optimisation problem

Life is an evolving process that started with entities retaining their complexity and replicating (information). It gradually grew more complex via Darwinian evolution (see Chapter (19)). Intelligent agents started as entities collecting information about their environment from sensors and then processing this information to decide how to act back on their environment. More formally, it is a system that perceives its environment and takes actions which maximise its chances of success (see Russell et al. [2009]). Scientists call it reinforcement learning. So, life can be described as a system made of:

- Software (made of bits): They are algorithms used to process information from sensors and decide what to do.
- Hardware (made of atoms): It is the matter.

Tegmark [2018] classified life forms into three levels of sophistication: life 1.0, 2.0 and 3.0.

- Life 1.0 is unable to redesign its hardware and software during its lifetime. They are determined by its DNA and modified via genetic evolution.
- Life 2.0 can redesign much of its software through learning, but the hardware is evolved across generations.
- Life 3.0 can redesign both its software and hardware.

Goal-oriented behaviour originate from the laws of physics. In general, nature behaves by optimising some quantities (see Chapter 18). For instance, from the second law of thermodynamics, it maximise entropy [2]. That is, nature maximise death and destruction (perfect uniformity). Dissipation is the act of transforming energy from some initial form to some final form with less capacity to perform mechanical work. This process produces entropy. Thus, dissipation means that total entropy must increase. For non-isolated systems such as organisms, they may lose entropy provided that their environment's entropy increases by at least that amount so that the total entropy increases.
On the other hand, we have seen above that the main goal for life was replication. Hence, replication is a subgoal that helps accomplish the fundamental goal of dissipation. Over time, other instrumental goals developed and genetically evolved with a life of their own. However, human beings are not rational agents making idealised decisions by optimising some utility functions. Rather, they have bounded rationality due to limited cognitive resources. As such they use heuristic simplifications (rules of thumb) to decide what to pursue and avoid (see Shleifer et al. [1997], Barberis et al. [1998]). These are feelings guiding our decision process, which may not always be appropriate (sub-optimums). Thus, humans do not have a single well-defined goal.

1.2.2.2 Defining intelligence

1.2.2.2.1 Human intelligence The word intelligence derives from the Latin nouns intelligentia or intellectus, meaning to comprehend or perceive. In the Middle Ages it was strongly linked to the metaphysical and cosmological theories of teleological scholasticism, before being related to the word understanding by the early modern philosophers such as Bacon, Hobbes, Locke, and Hume.
Nowadays, scientists put forward the idea that there is no consensus on defining intelligence (see Legg et al. [2007a]). Rather, there are competing definitions, including the capacity for logic (reasoning), understanding, planning, emotional knowledge, self-awareness, creativity, problem solving and learning. Nonetheless, according to a collective of researchers, it is accepted that intelligence reflects a broader and deeper capability for comprehending our surroundings, making sense of things (see Gottfredson [1997]). It is the ability to deal with cognitive [3] complexity (see

[2] The entropy is given by $S = k_B \ln \Omega$, where Ω is the number of microstates and k_B is the Boltzmann constant. It measures how messy things are (randomness of a system).

[3] Cognition is the mental action or process of acquiring knowledge and understanding through thought, experience, and the senses. It encompasses processes such as attention, the formation of knowledge, memory, judgement and evaluation, reasoning, decision making, comprehension and production of language. Cognitive processes use existing knowledge and generate new knowledge.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Gottfredson [1998]). Going further, human intelligence is the intellectual power of humans, which is marked by complex cognitive feats and high levels of motivation and self-awareness (see Tirri et al. [2011]). Another definition of intelligence involves the creation and use of persistent memories as opposed to computation that does not involve learning. The general factor of intelligence, called g-factor, is a psychometric construct summarising the correlations observed between an individual's scores on a wide range of cognitive abilities (see Kamphaus [2005]).

1.2.2.2.2 Noisy behaviour As a result, in order for a machine to replicate an anthropologic intelligence, one would need to understand humans driving goals by building a detailed model of the world. It would involve encoding arbitrary systems of goals and ethical principles into the machine. Further, machines would need to figure out the specific system that best matches the behaviour observed. That is, learning human behaviour. This is called inverse reinforcement learning, which infer the reward function from the observed behaviour of an expert (see Ng et al. [2000]). However, as we will discuss below, human beings are irrational agents guided by feelings. It translates into noisy behaviour which would need to be filtered out to get a chance of inferring the true behaviour of a human being by inverting RL. Inferring humans behaviour is probably more difficult than predicting the stock market.

1.2.2.2.3 Machine intelligence It seems that to avoid the difficulties associated with human intelligence, scientists in information technology (IT) reduced intelligence to the sole action of solving tasks. For instance, Legg et al. [2007b] gave a definition of machine intelligence as a synthesis of over 70 definitions from psychology, philosophy, and AI researchers: intelligence measures an agent's ability to achieve goals in a wide range of environments. As discussed above, one way of quantifying intelligence is trough comparison: assuming two programs, one of them is more intelligent than the other one if it is at least as good at accomplishing all goals (tasks), and strictly better at at least one task. However, this is only true if a goal is objectively quantifiable.
In order to quantify the complexity of goals, Moravec [1998] argued that computers are universal machines with potential extending universally over a boundless expanse of tasks. On the other hand, human potentials are strong in areas long important for survival, but weak in things far removed.
Expanding on the complexity of goals, artificial general intelligence (AGI) developed as maximally broad: it is the ability to accomplish any goal at least as well as humans (see Goertzel et al. [2007]). In order to take a broad and inclusive view, Tegmark [2018] defined intelligence as follows:

$$\text{intelligence} \;=\; \text{ability to accomplish complex goals}$$

leading to several types of intelligence (narrow and broad intelligence).

1.2.2.3 Accomplishing complex goals

1.2.2.3.1 Solving tasks From the above discussion, intelligence has been reduced to solving tasks. Scientists focussed on what AI does, not how it subjectively feels. Ultimately, intelligence is all about information and computation to perform tasks. They consider a physics approach where everything is simply matter and energy moving around. The main idea is to understand the arrangement of particles producing intelligence. It leads to the following claim:

***Claim* 1** *Following the laws of physics, matter can be rearranged to remember, compute, and learn.*

Computation is defined as a transformation of one memory state into another one. There exists several ways to transform information, such as supervised learning and reinforcement learning. For simplicity of exposition we focus on the former.
Information is the input transformed via a function to produce an output, denoted $\widehat{Y} = F(X)$. For instance, the NAND gate takes two bits, A and B, as inputs and computes one bit C as output based on the rule that $C = 0$ if $A = B = 1$ and $C = 1$ otherwise. The function F represents the arrangement (or mapping) of particles in space and time, and $\widehat{Y}$ is the pattern generated. Here, F is the NAND gate, but that function can be more or less complex, even exhibiting dynamics. In general, a system evolves according to the law of physics. Scientists rely on Theorem

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(5.1.1), which states that the NAND gates are universal, since we can implement any well-defined function Y simply by connecting together NAND gates. The latter are called computronium (see Margolus [1987]), and there are plenty of other substances sharing this property. For instance, neural networks also act as computronium and can implement arbitrary computations. Simply put, matter if the function F which must rearrange itself to get better and better at computing the desired function $Y = f(X)$ by obeying the law of physics. The process of modifying F until $\widehat{Y}$ best approximates Y is called the learning process. It is denoted by $\|\widehat{Y} - Y\|$. It is called training in neural networks and requires a very large amount of data.
We see that supervised learning requires knowing a well desired function $Y = f(X)$ to train the algorithm. However, there are innovative tasks requiring cognitive processes, such as research and artistic composition, where the function Y, representing new knowledge and discovery, is not known in advance and can not be inferred from past information.

1.2.2.3.2 Replicating human intelligence It has been argued that AI will keep progressing until it matches human abilities on most tasks. However, as discussed above, human beings are irrational agents guided by their feelings to make sub-optimal decisions. This process incorporate noise in the system, which is not captured by learning algorithms (the universal theorem does not apply). Thus, humans do not have a single well-defined goal, but a plethora of evolving goals.
One very good example of this phenomena is predicting financial market returns. Behavioural finance developped as a way of understanding noise from irrational investors. In perfect markets (complete), rational investors would make forecast that correctly incorporate all available information, leaving no room for systematic forecasting errors. Behavioural economics and behavioural finance have challenged this paradigm (see Barberis et al. [1998] [2002]). As fully rational behaviour requires quite complex calculations (optimisation tools), an alternative idea developed suggesting that investors exhibit bounded rationality, rationality that is limited by their cognitive resources and observational powers. Psychological biases predict specific systematic deviations from rationality causing mispricings. The main biases are heuristic simplifications, such as rules of thumb, mental shortcuts, attention and memory biases, representativeness, conservatism, and/or self-deception such as overconfidence, overoptimism, biased self-attribution, confirmation bias, hindsight bias. The best known market-level mispricings are speculative bubbles, and the best known relative mispricings are value and momentum effects. Even though it is argued that rational traders will view any such mispricings as trading opportunities, which when realised, will undo any price impact irrational traders might have, it did not happen in practice since these strategies are risky and costly (see Shleifer et al. [1997]). In fact, any observed predictability pattern can be interpreted to reflect either irrational mispricing, or rational time-varying risk premia (rational learning about structural changes).

1.2.2.4 Accounting for nonlinear information

So far, and as this book illustrates, machine learning models are not mathematical models but statistical models. Techniques are derived from statistical inference (see Chapter (11)) and probabilities (see Chapter (**??**)), and when it comes to sequential decision problems, the Markov process is assumed with transition probabilities statistically estimated (see Chapter (17)). Most mathematical proof for convergence of ML models assume infinite data, which is equivalent to knowing total information in probability. As a result, the performances of machine learning methods for data mining are heavily dependent on the choice of data representation on which they are applied (see Section (3.3.8)).
Further, when modelling human behaviour, one must face the fact that information arrives in a lumpy, discontinuous manner. Further, new information may contribute to increased levels of uncertainty, rather than increased levels of knowledge. This is because humans do not have a single well-defined goal. Thus, one must account for the possibility of nonlinear reaction to information.
When it comes to modelling human behaviour, mathematical finance is probably the field that put the most effort in developing prediction models. While originally modelled as random variables (efficient market hypothesis), Mandelbrot [1960] [1963] [1963a] described the financial market as a system with fat tails, stable distributions and persistence. Hence, long memory effects severely inhibited the validity of econometric models, explaining the poor record economists had in forecasting market returns. We now know that financial time series are multifractal, thus exhibiting non-Gaussian distribution, the presence of extreme values (outliers), and long-range dependent dynamics

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(see Bloch [2014]).
Some recent research on complex networks has shown that several real networks (describing human behaviour) were fractal, and even multifractal.

1.2.3 Description

Similarly to the intelligent agent described above, artificial intelligence (AI) analyses its environment and takes actions that maximises its chance of success (see Russell et al. [2009]). Many AI algorithms are capable of learning from data, leading to machine learning (ML). The main goal of ML is to create technology allowing computers and machines to function in an intelligent manner. That is, ML can give computers the ability to learn without being explicitly programmed (see A. Samuel [1959]). It focuses on designing, developing and analysing methods giving a computer the ability to follow a systematic process leading to producing results in difficult tasks that classical algorithms could not produce.
In practice, machine learning explores the study and construction of algorithms that can learn from and make predictions on data (see Kohavi et al. [1998]). Such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs (see Bishop [2006]). However, learning algorithms work on the basis that strategies, algorithms, and inferences that worked well in the past are likely to continue working well in the future. These inferences can be obvious, or they can be nuanced. Learners also work on the basis of "Occam's razor": The simplest theory that explains the data is the likeliest. Therefore, to be successful, a learner must be designed such that it prefers simpler theories to complex theories, except in cases where the complex theory is proven substantially better.

Machine learning (ML) is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible. The traditional problems (or goals) of ML research include reasoning, knowledge representation, planning, learning, natural language processing, perception and the ability to move and manipulate objects (see Russell et al. [2009]). General intelligence is among the field's long-term goals. Machine learning is closely related to statistical methods, computational intelligence, and traditional symbolic AI. Many tools are used in ML, including versions of search and mathematical optimisation, artificial neural networks, and methods based on statistics, probability and economics. The ML field draws upon computer science, information engineering, mathematics, psychology, linguistics, philosophy, and many other fields.

1.2.3.1 Knowledge representation

Knowledge representation and knowledge engineering are central to classical ML research. Davis [2015] defined knowledge representation as the study of how the beliefs, intentions, and value judgements of an intelligent agent can be expressed in a transparent, symbolic notation suitable for automated reasoning. For instance, some expert systems attempt to gather together explicit knowledge possessed by experts in some narrow domain. Further, some projects seek to gather the common-sense knowledge (such as CycL (AI project focussing on implicit representation)) known to the average person into a database containing extensive knowledge about the world. Some examples are: objects, properties, categories and relations between objects; situations, events, states and time; causes and effects; knowledge about knowledge (what we know about what other people know); and many other, less well researched domains.
Knowledge representation and reasoning plays a central role in Artificial Intelligence. One way of addressing this problem is to use some form of logic (see Nebel [2015]). For example, rule-based approaches are good for representing and utilising complex logic. A representation of "what exists" is an ontology: the set of objects, relations, concepts, and properties formally described so that software agents can interpret them. The semantics of these are captured as description logic concepts, roles, and individuals, and typically implemented as classes, properties, and individuals in the Web Ontology Language. The most general ontologies are called upper ontologies, which attempt to provide a foundation for all other knowledge by acting as mediators between domain ontologies that cover specific knowledge about a particular knowledge domain (field of interest or area of concern). Such formal knowledge representations can be used in content-based indexing and retrieval, scene interpretation, clinical decision support, knowledge discovery

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(mining interesting and actionable inferences from large databases), and other areas.
Among the most difficult problems in knowledge representation are:

- Default reasoning and the qualification problem: For any common-sense rule that ML researchers care to represent, there tend to be a huge number of exceptions. Almost nothing is simply true or false in the way that abstract logic requires.
- The breadth of common-sense knowledge: The number of atomic facts that the average person knows is very large. Research projects that attempt to build a complete knowledge base of common-sense knowledge require enormous amounts of laborious ontological engineering.
- The subsymbolic form of some common-sense knowledge: Much of what people know is not represented as facts or statements that they could express verbally. These are non-conscious and sub-symbolic intuitions or tendencies in the human brain.

1.2.3.2 The learning

1.2.3.2.1 Online and offline learning In computer science we distinguish between two types of learning:

1. Online: An online algorithm is one that can process its input piece-by-piece in a serial fashion, that is, in the order that the input is fed to the algorithm, without having the entire input available from the start. In operations research, the area in which online algorithms are developed is called online optimisation.

2. Offline: In contrast, an offline algorithm is given the whole problem data from the beginning and is required to output an answer which solves the problem at hand.

Since an online algorithm does not know the whole input, it is forced to make decisions that may later turn out not to be optimal. Thus, the study of online algorithms has focused on the quality of decision-making that is possible in this setting. Competitive analysis formalises this idea by comparing the relative performance of an online and offline algorithm for the same problem instance (see Borodin et al. [1998]). Specifically, the competitive ratio of an algorithm, is defined as the worst-case ratio of its cost divided by the optimal cost, over all possible inputs. The competitive ratio of an online problem is the best competitive ratio achieved by an online algorithm. That is, the competitive ratio of an algorithm gives a measure on the quality of solutions produced by this algorithm, while the competitive ratio of a problem shows the importance of knowing the future for this problem.
As an example, we consider the sorting algorithms with the following features:

- Selection sort: It repeatedly selects the minimum element from the unsorted remainder and places it at the front, which requires access to the entire input; it is thus an offline algorithm.
- Insertion sort: It considers one input element per iteration and produces a partial solution without considering future elements. Thus insertion sort is an online algorithm producing the optimum result, that is, a correctly sorted list.

1.2.3.2.2 The types of learning Machine learning tasks are typically classified into several broad categories:

- Supervised learning: The computer is presented with example inputs and their desired outputs, given by a teacher, and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback. Examples are classification, regression, object detection, semantic segmentation, images, captioning, among others.
- Semi-supervised learning: The computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Active learning: The computer can only obtain training labels for a limited set of instances (based on a budget), and also has to optimise its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labelling.
- Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning). Examples are clustering, dimensionality reduction, feature learning, density estimation, among others.
- Reinforcement learning: Data (in form of rewards and punishments) are given only as feedback to the program's actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent. The goal is to learn how to take actions in order to maximise reward. RL can be integrated with artificial neural networks to obtain high-quality generalisation, speeding up the learning process.
- Sequential decision making: Markov decision problems are sequential decision-making problems where decisions made in each state affect the trajectory of the states visited by the system over a particular time.

1.2.3.2.3 Some tasks Another categorisation of machine learning tasks arises when one considers the desired output of a machine-learned system:

- In classification, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised way. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam", represented by the Boolean values true and false (see Section (2.1.1)).
- In regression, also a supervised problem, the outputs are continuous rather than discrete.
- In clustering, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.
- Density estimation finds the distribution of inputs in some space.
- Dimensionality reduction simplifies inputs by mapping them into a lower-dimensional space. Topic modelling is a related problem, where a program is given a list of human language documents and is tasked to find out which documents cover similar topics.

Regression is a technique used to interpret experimental data which consists in finding the coefficients of a predefined function such that the function best fits the data. However, if the fit is not good, the experimenter has to keep trying different functions until a good model for the data is found. In many domains there is a strong tradition of only using linear or quadratic models, even though it is possible that the data would be better fit by some other model. Symbolic regression, instead, consists in finding a general function (with its coefficients) that fits the given data points. Since ML does not assume a priori a particular structure for the resulting function, it is well suited to this sort of discovery task.

1.2.3.3 Attention mechanisms

Attention is the behavioural and cognitive process of selectively concentrating on a discrete aspect of information, whether considered subjective or objective, while ignoring other perceivable information (see Zhaoping [2014]). Focalisation, concentration, of consciousness are of its essence. Attention has also been described as the allocation of limited cognitive processing resources. It is manifested by an attentional bottleneck, in term of the amount of data the brain can process each second. For example, in human vision, only less than 1% of the visual input data (at around one megabyte per second) can enter the bottleneck, leading to inattentional blindness.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In the context of machine learning, **attention** is a technique that mimics the effect of the cognitive process of attention in organisms. They are used in a wide variety of machine learning models, including in natural language processing (NLP) and computer vision. The attention mechanisms emerged naturally from problems that deal with time-varying data (sequences). They are used in Transformer networks to achieve their expressive power. Computer vision systems based on convolutional neural networks can also benefit from attention mechanisms. The two commonest attention techniqes used are dot-product attention (see Luong et al. [2015]), which uses the dot product between vectors to determine attention, and multi-head attention, which combines several different attention mechanisms to direct the overall attention of a network or sub-network. In computer vision, Larochelle et al. [2010] proposed that by looking at different parts of the image (glimpses), we can learn to accumulate information about a shape and classify the image accordingly. The same principle was later extended to sequences in NLP. We can look at all the different words at the same time and learn to "pay attention" to the correct ones depending on the task at hand.

Very deep neural networks already learn a form of implicit attention (see Mnih et al. [2014]). Deep networks are very rich function approximators. As such, they tend to ignore parts of the input and focus on others. For instance, when working on human pose estimation, the network will be more sensitive to the pixels of the human body. Misra et al. [2016] observed that many activation units showed a preference for human body parts and pose.

One way of understanding implicit attention is by considering the partial derivatives of the network with respect to the input, that is, the Jacobian matrix. By asking the network to "weigh" its sensitivity to the input based on memory from previous inputs, we introduce explicit attention. From now on, we will refer to this as attention. When attention is parametrised by differentiable functions, this is termed as soft attention. Soft attention means that the function varies smoothly over its domain and, as a result, it is differentiable. Historically, there is another concept called hard attention. In general, hard means that it can be described by discrete variables while soft attention is described by continuous variables. In other words, hard attention replaces a deterministic method with a stochastic sampling model. Since hard attention is non-differentiable, we cannot use the standard gradient descent. That is why we need to train them using Reinforcement Learning (RL) techniques such as policy gradients (see Section (17.1.4)) and the REINFORCE algorithm.

1.2.3.4 Control architectures

We expend on control system introduced in Section (1.1.3.2) by discussing its architecture.

1.2.3.4.1 Some terminologies

- A computer is a device that can be instructed to carry out sequences of arithmetic or logical operations automatically via computer programming. Modern computers have the ability to follow generalised sets of operations, called programs. These programs enable computers to perform an extremely wide range of tasks. A complete computer including the hardware, the operating system (main software), and peripheral equipment required and used for full operation can be referred to as a computer system.
- A control system manages, commands, directs, or regulates the behaviour of other devices or systems using control loops. It can range from a single home heating controller using a thermostat controlling a domestic boiler to large Industrial control systems which are used for controlling processes or machines.
- A state is a description of the status of a system that is waiting to execute a transition.
- A transition is a set of actions to be executed when a condition is fulfilled or when an event is received.

In some finite-state machine representations, it is possible to associate actions with a state:

- an entry action: performed when entering the state, and

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- an exit action: performed when exiting the state.

A state diagram is a type of diagram used in computer science and related fields to describe the behaviour of systems. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics. State diagrams can be used to graphically represent finite state machines. Another possible representation is the State transition table. A classic form of state diagram for a finite state machine or finite automaton (FA) is a directed graph. In graph theory (see Section (13.1)), a directed graph (or digraph) is a graph that is made up of a set of vertices connected by edges, where the edges have a direction associated with them. Directed graphs without loops are addressed as simple directed graphs, while directed graphs with loops are addressed as loop-digraphs. See Figure (1.1).

Figure 1.1: Simple directed graph

1.2.3.4.2 Finite-state machine A finite-state machine (FSM) is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some external inputs. The change from one state to another is called a transition. An FSM is defined by a list of its states, its initial state, and the conditions for each transition. Finite state machines are of two types: deterministic finite state machines and non-deterministic finite state machines. A deterministic finite-state machine can be constructed equivalent to any non-deterministic one.
The behaviour of state machines can be observed in many devices in modern society that perform a predetermined sequence of actions depending on a sequence of events with which they are presented. Simple examples are vending machines, which dispense products when the proper combination of coins is deposited, elevators, whose sequence of stops is determined by the floors requested by riders, traffic lights, which change sequence when cars are waiting, and combination locks, which require the input of combination numbers in the proper order.
Finite state machines can be subdivided into

- Transducers: They generate output based on a given input and/or a state using actions. They are used for control applications and in the field of computational linguistics.
- Acceptors: They produce binary output, indicating whether or not the received input is accepted. Each state of an FSM is either "accepting" or "not accepting". Once all input has been received, if the current state is an accepting state, the input is accepted, otherwise it is rejected. As a rule, input is a sequence of symbols (characters); actions are not used.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Classifiers: It is a generalisation of a finite state machine that, similar to an acceptor, produces a single output on termination but has more than two terminal states.
- Sequencers (or generators): They are a subclass of the acceptor and transducer types that have a single-letter input alphabet. They produce only one sequence which can be seen as an output sequence of acceptor or transducer outputs.

There are other sets of semantics available to represent state machines. For example, there are tools for modelling and designing logic for embedded controllers. They combine hierarchical state machines (which usually have more than one current state), flow graphs, and truth tables into one language, resulting in a different formalism and set of semantics. These charts, like Harel's original state machines, support hierarchically nested states, orthogonal regions, state actions, and transition actions.
Optimising an FSM means finding a machine with the minimum number of states that performs the same function. The fastest known algorithm doing this is the Hopcroft minimisation algorithm.

1.2.3.4.3 Behaviour tree A behaviour tree (BT) is a mathematical model of plan execution used in computer science, robotics, control systems and video games. They describe switchings between a finite set of tasks in a modular fashion. Their strength comes from their ability to create very complex tasks composed of simple tasks, without worrying how the simple tasks are implemented. BTs present some similarities to hierarchical state machines with the key difference that the main building block of a behaviour is a task rather than a state. Its ease of human understanding make BTs less error prone and very popular in the game developer community. BTs have been shown to generalise several other control architectures such as

- The subsumption architecture
- The teleo-reactive programs
- Sequential behaviour decompositions
- Decision tree

BT is graphically represented as a directed tree (see Section (13.1)) in which the nodes are classified as root, control flow nodes, or execution nodes (tasks) (see Colledanchise et al. [2014]). For each pair of connected nodes the outgoing node is called parent and the incoming node is called child. The root has no parents and exactly one child, the control flow nodes have one parent and at least one child, and the execution nodes have one parent and no children. Graphically, the children of a control flow node are placed below it, ordered from left to right.
We briefly describe some of these nodes:

- Control flow node: It is used to control the subtasks of which it is composed. A control flow node may be either a selector (fallback) node or a sequence node. They run each of their subtasks in turn. When a subtask is completed and returns its status (success or failure), the control flow node decides whether to execute the next subtask or not.
- Selector (fallback) nodes: They are used to find and execute the first child that does not fail. A fallback node will return immediately with a status code of success or running when one of its children returns success or running. The children are ticked in order of importance, from left to right. See Figure (1.2)
- Sequence nodes: They are used to find and execute the first child that has not yet succeeded. A sequence node will return immediately with a status code of failure or running when one of its children returns failure or running. The children are ticked in order, from left to right. See Figure (1.3)

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 1.2: BT selector

Figure 1.3: BT sequence

We now discuss the mathematical representation of the state-space model in the case of tree architecture (see Appendix (23.4.3)). In order to apply control theory tools to the analysis of behaviour trees (BTs), BT can be defined as three-tuple

$$T_i = \{f_i, r_i, \Delta t\}$$

where $i \in \mathbb{N}$ is the index of the tree, $f_i : \mathbb{R}^n \to \mathbb{R}^n$ is a vector field representing the right hand side of an ordinary difference equation, Δt is a time step and $r_i : \mathbb{R}^n \to \{R_i, S_i, F_i\}$ is the return status, that can be equal to either Running R_i, Success S_i, or Failure F_i. See Figure (1.4).
The execution of a BT is described by the following standard ordinary difference equations:

$$\begin{aligned} x_{k+t}(t_{k+1}) &= f_i\big(x_{k+}(t_k)\big) \\ t_{k+1} &= t_k + \Delta t \end{aligned}$$

where $k \in \mathbb{N}$ represents the discrete time, and $x \in \mathbb{R}^n$ is the state-space of the system modelled by the behaviour tree.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 1.4: (a) behaviour tree composition (b) corresponding vector field

1.2.3.5 Planning

Intelligent agents must be able to set goals and achieve them. As such, they need a way of visualising the future, getting a representation of the state of the world, and, they need be able to make predictions about how their actions will change it. As a result, agents must make choices that maximise the utility (or value) of available choices.
In classical planning problems, the agent can assume that it is the only system acting in the world, allowing the agent to be certain of the consequences of its actions. However, if the agent is not the only actor, then it requires that the agent can reason under uncertainty. This calls for an agent that can not only assess its environment and make predictions, but also evaluate its predictions and adapt based on its assessment.
Multi-agent planning uses the cooperation and competition of many agents to achieve a given goal. Planning algorithms search through trees of goals and subgoals, attempting to find a path to a target goal, a process called means-ends analysis. More generally, such emergent behaviours use evolutionary algorithms and swarm intelligence (see Section (19.1)).

1.2.3.5.1 Description Automated planning and scheduling or AI planning, is a branch of artificial intelligence that concerns the realisation of strategies or action sequences, typically for execution by intelligent agents, autonomous robots and unmanned vehicles (see Ghallab et al. [2004]). Unlike classical control and classification problems, the solutions are complex and must be discovered and optimised in multidimensional space (see Chapter (19)). Planning is also related to decision theory.
We distinguishes two types of environments:

1. Known environments: In known environments with available models, planning can be done offline. Solutions can be found and evaluated prior to execution.

2. Unknown environments: In dynamically unknown environments, the strategy often needs to be revised online. Models and policies must be adapted. Solutions usually resort to iterative trial and error processes commonly seen in artificial intelligence. These include dynamic programming, reinforcement learning (see Chapter (17)) and combinatorial optimisation (see Section (21.2)). Languages used to describe planning and scheduling are often called action languages.

Given a description of the possible initial states of the world, a description of the desired goals, and a description of a set of possible actions, the planning problem is to synthesise a plan that is guaranteed (when applied to any of the initial states) to generate a state which contains the desired goals (such a state is called a goal state).
The difficulty of planning depends on the simplifying assumptions employed. Several classes of planning problems can be identified based on the properties the problems in several dimensions. Hence, we must assess these properties by finding out:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- if the actions are deterministic or nondeterministic. In the latter we need to find out if the associated probabilities are available.
- if the state variables are discrete or continuous. In the former, do they have only a finite number of possible values?
- if the current state can be observed unambiguously. There can be full observability and partial observability.
- the number of initial states. Are they finite or arbitrarily many?
- if the actions have a duration.
- if several actions can be taken concurrently, or only one action is possible at a time.
- if the objective of a plan is to reach a designated goal state, or to maximise a reward function.
- if there is only one agent or there are several agents. Are the agents cooperative or selfish? Do all of the agents construct their own plans separately, or are the plans constructed centrally for all agents?

The simplest possible planning problem, called the Classical Planning Problem, is determined by:

- a unique known initial state,
- durationless actions,
- deterministic actions, which can be taken only one at a time,
- and a single agent.

Since the initial state is known unambiguously, and all actions are deterministic, the state of the world after any sequence of actions can be accurately predicted, and the question of observability is irrelevant for classical planning. Further, plans can be defined as sequences of actions, because it is always known in advance which actions will be needed.
In the case of non-deterministic actions, or other events outside the control of the agent, the possible executions form a tree, and plans have to determine the appropriate actions for every node of the tree. Some solutions include discrete-time Markov decision processes (MDP), which are planning problems with:

- durationless actions,
- non-deterministic actions with probabilities,
- full observability,
- maximisation of a reward function,
- and a single agent.

When full observability is replaced by partial observability, planning corresponds to partially observable Markov decision process (POMDP). If there are more than one agent, we have multi-agent planning, which is closely related to game theory.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.2.3.5.2 Domain modelling Domain independent planners typically input a domain model (a description of a set of possible actions which model the domain) as well as the specific problem to be solved specified by the initial state and goal. Typical examples of domains are block stacking, logistics, workflow management, and robot task planning. Hence a single domain independent planner can be used to solve planning problems in all these various domains. On the other hand, a route planner is typical of a domain specific planner. Language based on state variables is used to represent planning domains and specific planning problems, such as STRIPS and PDDL for Classical Planning. Each possible state of the world is an assignment of values to the state variables, and actions determine how the values of the state variables change when that action is taken. Since a set of state variables induce a state space that has a size that is exponential in the set, planning, similarly to many other computational problems, suffers from the curse of dimensionality and the combinatorial explosion. Alternative languages include hierarchical task networks, in which a set of tasks is given, and each task can be either realised by a primitive action or decomposed into a set of other tasks. This does not necessarily involve state variables, although in more realistic applications state variables simplify the description of task networks.

1.2.3.5.3 Some algorithms Some algorithms for planning are as follows:

- Classical planning: Examples of which are
  - Forward chaining state space search, possibly enhanced with heuristics: It is one of the two main methods of reasoning when using an inference engine and can be described logically as repeated application of modus ponens. Forward chaining is a popular implementation strategy for expert systems, business and production rule systems. The opposite of forward chaining is backward chaining.
  - Backward chaining search, possibly enhanced by the use of state constraints (see STRIPS, graphplan)
  - Partial-order planning: It is an approach to automated planning that maintains a partial ordering between actions and only commits ordering between actions when forced to. By contrast, total-order planning maintains a total ordering between all actions at every stage of planning. Given a problem in which some sequence of actions is required in order to achieve a goal, a partial-order plan specifies all actions that need to be taken, but specifies an ordering between actions only where necessary.
- Reduction to other problems: Examples of which are
  - reduction to the propositional satisfiability problem (satplan).
  - reduction to Model checking, both are essentially problems of traversing state spaces, and the classical planning problem corresponds to a subclass of model checking problems.
- Temporal planning: It can be solved with methods similar to classical planning. The main difference is that the definition of a state has to include information about the current absolute time and how far the execution of each active action has proceeded. Further, in planning with rational or real time, the state space may be infinite, unlike in classical planning or planning with integer time. Temporal planning is closely related to scheduling problems.
- Probabilistic planning: It can be solved with iterative methods such as value iteration and policy iteration, when the state space is sufficiently small. With partial observability, probabilistic planning is similarly solved with iterative methods, but using a representation of the value functions (VF) defined for the space of beliefs instead of states.
- Preference-based planning: The objective is not only to produce a plan but also to satisfy user-specified preferences. A difference to the more common reward-based planning, for example corresponding to MDPs, preferences do not necessarily have a precise numerical value.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Conditional planning: Deterministic planning is associated to hierarchical planning, which can be compared with an automatic generated behaviour tree. One disadvantage is that a normal behaviour tree is not so expressive like a computer program. That is, the notation of a behaviour graph contains action commands, but no loops or if-then-statements. Conditional planning overcomes the bottleneck and introduces an elaborated notation which is similar to a control flow [4], known from other programming languages like Pascal. It is very similar to program synthesis, that means a planner generates source-code which can be executed by an interpreter.
  A complicated plan containing if-then-statements has to do with uncertainty at runtime of a plan. The idea is that a plan can react to sensor signals which are unknown for the planner (see Karlsson [2001]). The planner generates two choices in advance. For example, if an object was detected, then action A is executed, if an object is missing, then action B is executed. One major advantage of conditional planning is the ability to handle partial plans. An agent is not forced to plan everything from start to finish but can divide the problem into chunks. This helps to reduce the state-space and solves much more complex problems (see Liu [2008]).

1.2.4 Some properties

1.2.4.1 Prediction

Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction, known as predictive analytics in commercial use. These analytical models allow researchers, data scientists, engineers, and analysts to produce reliable, repeatable decisions and results and uncover hidden insights through learning from historical relationships and trends in the data.
Machine learning is closely related to computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimisation, which delivers methods, theory and application domains to the field. Machine learning is sometimes conflated with data mining, where the latter subfield focuses more on exploratory data analysis and is known as unsupervised learning (see Friedman [1998]).
Machine learning and data mining often employ the same methods and overlap significantly, but while machine learning focuses on prediction, based on known properties learned from the training data, data mining focuses on the discovery of (previously) unknown properties in the data (this is the analysis step of knowledge discovery in databases). Data mining uses many machine learning methods, but with different goals; on the other hand, machine learning also employs data mining methods as unsupervised learning or as a preprocessing step to improve learner accuracy.

1.2.4.2 Optimisation

Machine learning also has intimate ties to optimisation: many learning problems are formulated as minimisation of some loss function on a training set of examples. Loss functions express the discrepancy between the predictions of the model being trained and the actual problem instances (for example, in classification, one wants to assign a label to instances, and models are trained to correctly predict the pre-assigned labels of a set of examples). The difference between the two fields arises from the goal of generalisation: while optimisation algorithms can minimise the loss on a training set, machine learning is concerned with minimising the loss on unseen samples (see Le Roux et al. [2012]).

1.2.4.3 Generalisation

One of the main objective of machine learning is to generalise from its experience (see Bishop [2006], Mohri et al. [2012]). Generalisation in this context is the ability of a learning machine to perform accurately on new, unseen examples/tasks after having experienced a learning data set. The training examples come from some generally unknown probability distribution (considered representative of the space of occurrences) and the learner has to build a general model about this space that enables it to produce sufficiently accurate predictions in new cases.

[4] Control flow is the order in which individual statements, instructions or function calls of an imperative program are executed or evaluated. The emphasis on explicit control flow distinguishes an imperative programming language from a declarative programming language.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The computational analysis of machine learning algorithms and their performance is a branch of theoretical computer science known as computational learning theory. Because training sets are finite and the future is uncertain, learning theory usually does not yield guarantees of the performance of algorithms. Instead, probabilistic bounds on the performance are quite common. The bias-variance decomposition is one way of quantifying generalisation error.

For the best performance in the context of generalisation, the complexity of the hypothesis should match the complexity of the function underlying the data. If the hypothesis is less complex than the function, then the model has underfit the data. However, if the complexity of the model is increased in response, then the training error should decrease. But if the hypothesis is too complex, then the model is subject to overfitting and generalisation will be poorer (see Alpaydin [2010]).

In addition to performance bounds, computational learning theorists study the time complexity and feasibility of learning. In computational learning theory, a computation is considered feasible if it can be done in polynomial time. There are two kinds of time complexity results. Positive results show that a certain class of functions can be learned in polynomial time. Negative results show that certain classes cannot be learned in polynomial time.

1.2.4.4 Limitation

Effective machine learning is difficult because finding patterns is hard and often not enough training data are available. As a result, many machine-learning programs often fail to deliver the expected value. Reasons for this are numerous: lack of (suitable) data, lack of access to the data, data bias, privacy problems, badly chosen tasks and algorithms, wrong tools and people, lack of resources, and evaluation problems.
Further, machine learning approaches can suffer from different data biases. For instance, a machine learning system trained on one current customers only may not be able to predict the needs of new customer groups that are not represented in the training data. In addition, when trained on man-made data, machine learning is likely to pick up the same constitutional and unconscious biases already present in society.

1.2.5 Processes and techniques

Several processes, techniques and methods can be applied to one or more types of machine learning algorithms to enhance their performance. Some of them are listed below:

- Feature learning: It is motivated by the fact that machine learning tasks such as classification often require input that is mathematically and computationally convenient to process, which is not the case in real word data. Thus, it attempts at preserving the information in their input but also transform it in a way that makes it useful. Consequently, it is often used as a pre-processing step before performing classification or predictions. This technique allows reconstruction of the inputs coming from the unknown data-generating distribution, while not being necessarily faithful to configurations that are implausible under that distribution. As such, it replaces manual feature engineering, and allows a machine to both learn the features and use them to perform a specific task.
  Feature learning can be either supervised or unsupervised (see Jurafsky et al. [2009]). In the former, features are learned using labelled input data. Examples include artificial neural networks, multilayer perceptrons, and supervised dictionary learning. In the latter, features are learned with unlabelled input data. Examples include dictionary learning, independent component analysis, autoencoders, matrix factorisation and various forms of clustering (see Coates et al. [2011]).
  For instance, Manifold learning algorithms learn features under the constraint that the learned representation is low-dimensional, while Sparse coding algorithms do so under the constraint that the learned representation is sparse, meaning that the mathematical model has many zeros. Multilinear subspace learning algorithms aim to learn low-dimensional representations directly from tensor representations for multidimensional data, without reshaping them into higher-dimensional vectors (see Lu et al. [2011]). Deep learning algorithms discover

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

multiple levels of representation, or a hierarchy of features, with higher-level, more abstract features defined in terms of (or generating) lower-level features. It has been argued that an intelligent machine is one that learns a representation that disentangles the underlying factors of variation that explain the observed data (see Bengio [2009]).

- Sparse dictionary learning: It is a feature learning method where a training example is represented as a linear combination of basis functions, and is assumed to be a sparse matrix. The method is strongly NP-hard and difficult to solve approximately (see Tillmann [2015]). A popular heuristic method for sparse dictionary learning is the K-SVD algorithm. It has been applied in several contexts such as classification, where the problem is to determine to which classes a previously unseen training example belongs. For a dictionary where each class has already been built, a new training example is associated with the class that is best sparsely represented by the corresponding dictionary. It has also been applied in image de-noising where the key idea is that a clean image patch can be sparsely represented by an image dictionary, but the noise can not (see Aharon et al. [2006]).

- Anomaly detection: In data mining, anomaly detection (or outlier detection) is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data. Typically, the anomalous items represent an issue such as bank fraud, a structural defect, medical problems or errors in a text. Anomalies are referred to as outliers, novelties, noise, deviations and exceptions (see Hodge [2004]). However, in the context of abuse and network intrusion detection, the interesting objects are often not rare objects, but unexpected bursts in activity. This pattern does not adhere to the common statistical definition of an outlier as a rare object, and many outlier detection methods (in particular, unsupervised algorithms) will fail on such data, unless it has been aggregated appropriately. Instead, a cluster analysis algorithm may be able to detect the micro-clusters formed by these patterns (see Dokas et al [2002]).
  Three broad categories of anomaly detection techniques exist (see Chandola et al. [2009]). (1) Unsupervised anomaly detection techniques detect anomalies in an unlabelled test data set under the assumption that the majority of the instances in the data set are normal, by looking for instances that seem to fit least to the remainder of the data set. (2) Supervised anomaly detection techniques require a data set that has been labelled as "normal" and "abnormal" and involves training a classifier (the key difference to many other statistical classification problems is the inherent unbalanced nature of outlier detection). (3) Semi-supervised anomaly detection techniques construct a model representing normal behaviour from a given normal training data set, and then test the likelihood of a test instance to be generated by the model.

- Decision tree learning: It uses a decision tree as a predictive model, which maps observations about an item to conclusions about the item's target value. Tree models where the target variable can take a discrete set of values are called classification trees. In that case, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data, but the resulting classification tree can be an input for decision making.

- Association rule learning: It is a general term for any machine learning method that identifies, learns, or evolves rules to store, manipulate or apply knowledge. It is a method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using some measure of "interestingness" (see Piatetsky-Shapiro [1991]). This rule-based approach generates new rules as it analyses more data. Assuming the set of data is large enough, the ultimate goal is to help a machine mimic the human brain's feature extraction and abstract association capabilities for data that has not been categorised.
  Rule-based machine learning approaches include learning classifier systems, association rule learning, and artificial immune systems. They are outlined as follows:

  - Based on the concept of strong rules, Agrawal et al. [1993] introduced association rules (AR) for discovering regularities between products in large-scale transaction data recorded by point-of-sale (POS) systems in supermarkets. Such information can be used as the basis for decisions about marketing activities such as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

promotional pricing or product placements. In addition to market basket analysis, association rules are employed today in application areas including Web usage mining, intrusion detection, continuous production, and bioinformatics.

- Learning classifier systems (LCS) are a family of rule-based machine learning algorithms combining a discovery component, typically a genetic algorithm (GA), with a learning component, performing either supervised learning, reinforcement learning, or unsupervised learning. They seek to identify a set of context-dependent rules that collectively store and apply knowledge in a piecewise manner in order to make predictions.
- Inductive logic programming (ILP) is an approach to rule-learning using logic programming (LP) as a uniform representation for input examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic program that entails all positive and no negative examples. Inductive programming (IP) is a related field that considers any kind of programming languages for representing hypotheses (and not only logic programming), such as functional programs. ILP is particularly useful in bioinformatics and natural language processing (NLP). Plotkin [1970] and Shapiro [1981] [1983] laid the initial theoretical foundation for inductive machine learning in a logical setting.

1.2.6 Some models

There is a long list of machine learning algorithms. Some of them are outlined below:

- Artificial neural networks: It is a learning algorithm that is vaguely inspired by biological neural networks that constitute animal brains. Such systems learn (progressively improve their ability) to do tasks by considering examples, generally without task-specific programming. Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Each connection (synapse) between neurons can transmit a signal to another neuron. The receiving (postsynaptic) neuron can process the signal(s) and then signal downstream neurons connected to it. Typically, neurons are organised in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times. Modern neural networks are non-linear statistical data modelling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.

- Deep learning: It consists of multiple hidden layers in an artificial neural network. More formally, deep learning (DL) is a class of machine learning algorithms that:

  - use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input.
  - learn in supervised (such as classification) and/or unsupervised (such as pattern analysis) manners.
  - learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.

  This approach tries to model the way the human brain processes light and sound into vision and hearing. Some successful applications of deep learning are computer vision and speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs.

- Support vector machines: They are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Bayesian networks: It is a probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform inference and learning.

- Representation learning: Several learning algorithms, mostly unsupervised learning algorithms, aim at discovering better representations of the inputs provided during training. Classical examples include principal components analysis and cluster analysis. Representation learning algorithms often attempt to preserve the information in their input but transform it in a way that makes it useful, often as a pre-processing step before performing classification or predictions, allowing reconstruction of the inputs coming from the unknown data generating distribution, while not being necessarily faithful for configurations that are implausible under that distribution.

- Rule-based machine learning: It is a general term for any machine learning method that identifies, learns, or evolves rules to store, manipulate or apply, knowledge. The defining characteristic of a rule-based machine learner is the identification and utilization of a set of relational rules that collectively represent the knowledge captured by the system. This is in contrast to other machine learners that commonly identify a singular model that can be universally applied to any instance in order to make a prediction. Rule-based machine learning approaches include learning classifier systems, association rule learning, and artificial immune systems.

Note, the neural network (NN) itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

1.2.7 Problems solving

Some methods and tools have been developed in ML for solving some of the most difficult problems in computer science. Some of these methods are

- Logic: It is used for knowledge representation and problem solving, but it can be applied to other problems as well (see Section (2.1)). Several different forms of logic are used in ML research. For instance, propositional logic involves truth functions such as "or" and "not". First-order logic adds quantifiers and predicates, and can express facts about objects, their properties, and their relations with each other. Fuzzy set theory assigns a "degree of truth" (between 0 and 1) to vague statements such as "Paul is old" (or rich, or tall, or hungry) that are too linguistically imprecise to be completely true or false. Fuzzy logic is successfully used in control systems to allow experts to contribute vague rules such as "if you are close to the destination station and moving fast, increase the pressure of the train's brake". These vague rules can then be numerically refined within the system. However, fuzzy logic fails to scale well in knowledge bases. Default logics, non-monotonic logics and circumscription are forms of logic designed to help with default reasoning and the qualification problem. Several extensions of logic have been designed to handle specific domains of knowledge, such as: description logics, situation calculus, event calculus and fluent calculus (for representing events and time), causal calculus, belief calculus and modal logics.

- Classifiers and statistical learning methods: Simple ML applications can be divided into two types: classifiers ("if shiny then diamond") and controllers ("if shiny then pick up"). Controllers do, however, also classify conditions before inferring actions, and therefore classification forms a central part of many ML systems. Classifiers are functions that use pattern matching to determine a closest match (see Section (3.3)). They can be tuned according to examples known as observations or patterns. In supervised learning, each pattern belongs to a certain predefined class. A class can be seen as a decision that has to be made. All the observations combined with their class labels are known as a data set. When a new observation is received, that observation is classified based on previous experience.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

A classifier can be trained in various ways: there are many statistical and machine learning approaches such as the decision tree (see Section (4.1)). Other widely used classifiers are the neural network, k-nearest neighbour algorithm, kernel methods such as the support vector machine (SVM), Gaussian mixture model, and the popular naive Bayes classifier. Classifier performance depends greatly on the characteristics of the data to be classified, such as the dataset size, distribution of samples across classes, the dimensionality, and the level of noise.

- Search and optimisation: Many learning algorithms use search algorithms based on optimisation (see Chapter (18)). For many problems, it is possible to begin the search with some form of a guess and then refine the guess incrementally until no more refinements can be made. These algorithms can be visualised as blind hill climbing: we begin the search at a random point on the landscape, and then, by jumps or steps, we keep moving our guess uphill, until we reach the top. Other optimisation algorithms are simulated annealing, beam search and random optimisation. Evolutionary computation uses a form of optimisation search (see Chapter (19)). For example, they may begin with a population of organisms (the guesses) and then allow them to mutate and recombine, selecting only the fittest to survive each generation (refining the guesses). Classic evolutionary algorithms include genetic algorithms, gene expression programming, and genetic programming.

- Probabilistic methods for uncertain reasoning: Many problems in reasoning, planning, learning, perception, and robotics require the agent to operate with incomplete or uncertain information. These problems can be solved by using methods from probability theory and economics (see Chapter (**??**)). Probabilistic algorithms can also be used for filtering, prediction, smoothing and finding explanations for streams of data. For instance, Bayesian networks are a very general tool that can be used for a large number of problems. Complicated graphs with diamonds or other loops (undirected cycles) can require a sophisticated method such as Markov Chain Monte Carlo, which spreads an ensemble of random walkers throughout the Bayesian network and attempts to converge to an assessment of the conditional probabilities. Precise mathematical tools have been developed that analyse how an agent can make choices and plan, using decision theory, decision analysis, and information value theory. These tools include models such as Markov decision processes, dynamic decision networks, game theory and mechanism design.

1.2.8 Applications to the industry

As discussed in Section (1.2), machine learning is about programming computers to learn and to improve automatically with experience (see A. Samuel [1959]). While computers can not yet learn as well as people, algorithms (see Section (1.1.2)) have been devised that are effective for certain types of learning tasks, and a theoretical understanding of learning has emerged. Computer programs developed, exhibiting useful types of learning, especially in speech recognition and data mining (see Mitchell [1997]). Further, new techniques combining elements of learning, evolution and adaptation from the field of Computational Intelligence developed, capable of improving the work process and adding value to the industry by solving key business problems.

As a result, artificial intelligence (AI) is becoming an emerging asset to the workforce, and machine learning (ML) is growing in its reputation among business leaders. The latter is found in multiple industries, transforming the way business and societies operate. It will allow society to redefine efficiency. Some of the vertical breakdowns are as follows:

- Enterprise intelligence
- Enterprise functions
- Autonomous systems
- Healthcare
- Technology stack

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Energy
- Event management
- Industry (agriculture, marketing, law, logistics, retail, finance, transport)

Even though segmenting market is qualitatively messy, it might clear up in time. No one knows what will take off, and this influences the behaviour of the industry. Nonetheless, Healthcare, Marketing and Finance consistently appear as areas of ML focus.

1.2.8.1 Defining AI strategy and production readiness

The aim of AI in the industry is to make more informed decisions, obtain information faster, automate processes, and enable delivery faster than a human mind. To do so, it is necessary to understand the business strategy, challenges, and opportunities faced by the company, client's data and analytics capabilities. Combining these with an understanding of state-of-the art AI, one can identify the most important AI initiatives for the company. The steps are as follows:

- Align business goals with AI strategy: AI priorities are derived from business ones. One should consider the business case for each business area as well as AI's relative importance to the case.
- History: Study the projects implemented in the past. Analyse the results, the challenges, and the lessons learned.
- Strategic AI priorities:
  - Identify priorities and define use cases.
  - Focus on cost-saving cases: optimisation of current business processes (business models, products, services, internal processes and functions).
  - Identify short-term AI priorities: demonstrate value, gain support and buy-in for a data and AI vision.
  - Investigate new data-driven business opportunities.
- Data due diligence:
  - Understand and evaluate data sources relevant for AI models: where it is generated and saved.
  - Check whether the different datasets can be merged on unique identifiers.
  - Formulate fast approaches to benchmarking different AI solutions.
- Solution architecture and technology:
  - Have an experienced data and solution architect take a critical look into the current technical architecture and define the target architecture and its development roadmap. It should follow the end-to-end use case logic accounting for data collection from operating systems, data warehouses, cloud environments, analytical environments, and business-interfacing systems. -
  - Operative systems and backend systems need to be an integral part of the data and solution architecture.
- Data vision and strategy:
  - Provide recommendation for data and tools: must be structured according to the FAIR principles (Findable, Accessible, Interoperable, Reusable).
  - Provide recommendation for architecture: follow the principle of build once – use many.
- Operational model:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

  - Business leaders should establish an AI steering group or include the data and AI development into the existing leadership team meetings.
  - Data and AI development budgets should be centralised.
  - Manage the change process.

- AI algorithms:
  - The portfolio of machine learning / AI algorithms should be FAIR. It requires the data science team to use common code repositories and standards.
  - Establish maintenance processes for the data and algorithm assets.

- Improve AI solutions: continuous improvement.

- Scale AI enterprise wide: AI perpetual innovation.

Once the AI strategy has been defined, we need to execute the first use case aiming at production readiness. The steps are as follows:

- Check feasibility for each use case:
  - Assess maturity within relevant categories such as organisational setup, competencies, technology landscape, data availability and quality.
  - Assess gaps that need to be addressed.

- AI algorithms:
  - Identify AI models for the use case.
  - Identify required data.

- Data availability and quality:
  - Find which datasets are the most important to have, and which ones are optional.
  - Find out whether the necessary data is being captured.
  - Check whether the different datasets you need can be merged.

- Technology and infrastructure: Set out the technology-related implications, challenges requirements. Identify systems, software and hardware needed.

- Time to implement: Estimate implementation time for each action.

- Risk: Assess the unknown factors that could derail the project.

- Estimate potential gains.

- Implementation:
  - Identify potential implementation challenges.
  - Define the key steps for delivering project.
  - Identify responsibilities for delivering each action.

- Data protection and privacy: relates to data collection, processing, and utilisation. The team for setting up the company-wide privacy policies should consist of a combination of business owner, privacy lawyers, and AI strategists.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Data and AI roles: need a multi-disciplinary AI team.
  - Business case owner: responsible for the AI use case.
  - AI strategist: translates the business vision and goals into data and AI requirements, oversees project execution, and ensures that project outcomes are taken into use by business processes.
  - Data scientists come in various forms, with different backgrounds.
  - Data engineers and data architects.
  - Platform engineers and solution architects.
- Change of management: define a clear policy of change of management.
- Measures of success:
  - Analyse what success looks like for this use case.
  - Identify the measures of success to track progress.

1.2.8.2 Some examples

1.2.8.2.1 Energy AI are poised to revolutionise the way utilities produce, transmit, and consume energy by powering the modern smart grid. That is, AI is becoming the brain for smart grids. It has the potential to deliver the active management that will be required for the grid of the future. Powerful intelligence are going to be able to balance grids, manage demand, negotiate actions, enable self-healing and facilitate a host of new products and services. It is also going to enable more efficient and effective utility operations by helping to analyse unstructured data which typically makes up to 80 percent of data in an organisation.
The applications in the three largest segments of use are: renewable management, demand management, infrastructure management. This is becoming a reality with the first ultrahigh-voltage (UHV) AC-DC power grid in China.
AI can instantly analyse real-time data. It can spot patterns and anomalies in datasets, allowing utilities to make on-the-spot decisions about how to best allocate energy resources. The deep learning algorithms are already revolutionising both the demand and supply for the energy economy:

1. Improve distributed generation management. The current system is not constructed to accommodate energy source diversification. For example, the rise of distributed generation is complicating supply and demand forces. When private users generate their own electricity from renewable sources such as wind and solar, utility companies can absorb the excess energy into the grid. This complicates supply and demand, however. AI can help utilities realize the next-generation grid through enhanced distributed resource management that automatically flows power through the grid to deliver more reliable energy and greater customer choice.
2. Asset optimisation. Utilities are developing algorithms based on industry intelligence that will predict the probability of failure. These algorithms take into account industry-wide early failure rates for equipment, creating a richer understanding of premature failure risks for enhanced asset maintenance, workflow, and portfolio management.
3. Outage management. Utilities are using analytics-validating models to predict and identify outages. Machine learning and device automation allow for better resource management, reducing downtime and improving reliability. Self-healing grids can automatically detect and address vulnerabilities, reducing the likelihood of outages.
4. Customer engagement. Utilities are mining data with the aid of machine learning to understand customer behaviour and service needs. Using this data, utilities can provide faster and more intuitive interactive customer service via voice response, personalisation, and service matching.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.2.8.2.2 Event management Event management is the application of project management to the creation and development of large-scale events such as festivals, conferences, ceremonies, weddings, formal parties, concerts, or conventions. It involves studying the brand, identifying its target audience, devising the event concept, and coordinating the technical aspects before actually launching the event.
We need to build an AI program builder or optimum management system that will identify in an optimum way the delegates, the location, the venue, the speakers. It should identify the hotels and make reservations. Organise meet up, discussions, dinner and other social activities.

AI technology can analyse all the touchpoints along the delegate journey, it can be an integral part in helping to drive a more personalised and curated experience. As a result of the data constructed during previous events and the insight gained, an event management system can:

- Provide a registration assistant to a comprehensive virtual concierge. Chatbots are used to give consistent responses to questions via programming. They can work with the hotel guest services to have breakfast ready, book a taxi and then have coffee ready for the delegate at the venue. They provide reminders for event sessions, eventually providing suggestions for what to do next. They can cutdown the time which is wasted at events such as standing in line for the bathroom.
- Chatbots also give event profs real-time insight into levels of attendee engagement and satisfaction. For example, what questions are being asked the most? are there any recurring problems attendees are having?
- A chatbot provides relief to the team, preventing them from spending their time answering the same questions over and over, and leaving them to deal with the more important issues and aspects of an event launch.
- Provide real-time recommendations for networking opportunities. AI learns from delegate's behaviour as you interact with it, so attendees receive better recommendations the more they use it. This also helps exhibitors get to know their potential customers, so they can plan time, space, and staffing for the greatest visibility. They can also use this data to personalise their offerings to attendees. In addition, it can be used to make merchandise recommendations.
- Have a system reading footfall and registration data at an event, and then anticipating what logistical requirements are needed at the venue in terms of beverages, food, and crowd control, as the event progresses in real time. Judgements are based on data input from previous events of similar sizes and requirements, as well as on-site feedback of demand for the services provided.

1.2.8.2.3 Pharmacogenomics Pharmacogenomics (pharmacology plus genomics) is the study of how an individual's genetic inheritance affects the body's response to drugs. It aims at developing rational means to optimise drug therapy, with respect to the patients genotype, to ensure maximum efficacy with minimal adverse effects. Such approaches promise the advent of personalised medicine, in which drugs and drug combinations are optimised for each individual's unique genetic makeup. The way a person responds to a drug (this includes both positive and negative reactions) is a complex trait that is influenced by many different genes. Thus, understanding an individual's genetic makeup is the key to creating personalised drugs with greater efficacy and safety. For example, pharmacogenomics can help patients with depression avoid side effects of psychiatric medications. While knowing all of the genes involved in drug response is a difficult task, people's genes show small variations (or changes) in their nucleotide (DNA base) content so that genetic testing for predicting drug response is now possible. The market for DNA testing is growing exponentially (for example 23andMe) New developments in this field will impact drug design in three main ways: (1) the interaction of the drug with its receptor binding site; (2) the absorption and distribution of the drug; (3) the elimination of the drug from the body. Pharmacogenomics is one piece of the puzzle. It is a tool to help the physician that can be coupled with information from the patient, like their symptoms and family history, to look more closely at one class of drug over another. This information together with the hormone level and genetic data form the patient profile.
The need for AI are as follows:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1. AI can hold in its DB information about every known illness and medicine in history. It can update the DB daily with finding of new researches, with medical statistics gathered from every linked-in clinic and hospital in the world.
2. It can be intimately familiar with one entire genome and one day-to-day medical history. But also with the genomes and medical histories of the patient's parents, siblings, cousins, neighbours and friends. It will know instantly whether one visited a tropical country recently, if one has recurring stomach infections, whether there have been cases of intestinal cancer in one's family or whether people all over town are complaining about diarrhoea.
3. AI will never be tired, hungry or sick and will be dedicated to every patient in a unique way. Patient can stay home and not waist time going to the GP.
4. CT scans and mammography exams are routinely checked by specialised algorithms, which provide doctors with a second opinion and detect tumours that doctors might miss.
5. By knowing its patient AI can take better decisions. Since it is all based on statistics, the size of the DB is the key of making accurate predictions.

1.2.8.2.4 Finance To illustrate some of the models and algorithms that we will present in this book we will draw examples from the Finance industry. This is because subjects such as Neural Networks, Swarm Intelligence, Fuzzy Systems and Evolutionary Computation can be applied to financial markets in a variety of ways such as predicting the future movement of stock's price or optimising a collection of investment assets (funds and portfolios). These techniques assume that there exist patterns in stock returns and that they can be exploited by analysis of the history of stock prices, returns, and other key indicators (see Schwager [1996]). With the fast increase of technology in computer science, new techniques can be applied to financial markets in view of developing applications capable of automatically managing a portfolio. Consequently, there is substantial interest and possible incentive in developing automated programs that would trade in the market much like a technical trader would, and have it relatively autonomous.
The commodity market is an example of such interest for AI technology. Some of the criteria affecting the commodity market are:

- Geopolitical risk
  - Trade wars: US and China lock horns
  - Protectionism: disruptive forces in metal markets
- New frontiers: national oil giants eye trading
- Technology: commodities 2.0
  - Blockchain
  - Digital processes replacing people, phones and paper trails could significantly reduce trading costs.
  - Data manipulation and interpretation. Tighter margins have made participants look closer at accessing faster data, instead of operating scale, to gain an advantage.
  - Fast access to data such as geospatial imagery, smart metering, or better visibility of tankers on water are increasingly giving traders an edge. Practitioners see data science playing an increasingly core role in multiple decision-making ranging from cross-regional arbitrage to daily stock management. These technologies will increasingly shape the markets in the coming years.
  - Sentiment Analysis: When Commodity Trading Meets Deep Learning. The automated analysis of textual data and its application in business analytics holds great promise for providing decision-makers with information from a sheer endless stream of news available online. Recent advances in computing have led to exciting new tools in the areas of Natural Language Processing, Sentiment Analysis and Machine Learning that can be used to make sense from an ever-growing number of online sources.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- ML models to forecast the commodity market.

1.3 Neural networks

1.3.1 An overview

1.3.1.1 A brief history

In the late 1940s, Hebb [1949] created a learning hypothesis based on the mechanism of neural plasticity that became known as Hebbian learning. It is unsupervised learning and evolved into models for long term potentiation. Researchers started applying these ideas to computational models in 1948 with Turing's B-type machines.
McCulloch and Pitts [1943] created a computational model for neural networks based on mathematics and algorithms called threshold logic. This model paved the way for neural network research to split into two approaches. One approach focused on biological processes in the brain, while the other focused on the application of neural networks to artificial intelligence. This work led to work on nerve networks and their link to finite automata (see Kleene [1956]).
Farley et al. [1954] first used computational machines, then called "calculators", to simulate a Hebbian network. Other neural network computational machines were created by Rochester et al. [1956].
Rosenblatt [1958] created the perceptron, an algorithm for pattern recognition. With mathematical notation, Rosenblatt described circuitry not in the basic perceptron, such as the exclusive-or circuit that could not be processed by neural networks at the time (see Werbos [1974]).
In 1959, a biological model proposed by Nobel laureates Hubel and Wiesel was based on their discovery of two types of cells in the primary visual cortex: simple cells and complex cells (see Hubel et al. [2005]). The first functional networks with many layers were published by Ivakhnenko et al. [1967], becoming the Group Method of Data Handling

1.3.1.2 A Description

Among the different networks existing, the artificial neural networks (ANNs) and the artificial recurrent neural networks (RNNs) are computational models designed by more or less detailed analogy with biological brain modules. The former is presented in Chapter (5) and the latter is introduced in Chapter (6). ANNs, formalised by McCulloch et al. [1943], are a class of generalised nonlinear nonparametric models inspired by studies of the human brain. They get their intelligence from learning process, giving them the capability of auto-adaptability, association, and memory to perform certain tasks. The backpropagation network, which is the most popular and the most widely implemented neural network in the financial industry, is based on a multi-layered feedforward topology with supervised learning (see Refenes et al. [1997]). The network is fully connected, with every node in the lower layer linked to every node in the next higher layer via weight values. The learning of the backpropagation neural network is based on an error minimisation procedure where the weights are modified according to an error function comparing the neural network output with the training targets. We evaluate the derivatives of the error function with respect to the weights which are used to compute the adjustment to be made to the weights. The simplest such techniques involves gradient descent. In the case of time series prediction, a set of input-target pairs is created, forming the training samples. Every time a new time series is generated, new observations are added to the set and the oldest ones are dropped out. Alternatively, artificial recurrent neural networks (RNNs) are a class of feedback artificial neural network architecture that uses iterative loops to store information, which is inspired by the cyclical connectivity of neurons in the brain. The existence of cycles allows RNNs to develop a self-sustained temporal activation dynamics along its recurrent connection pathways, even in the absence of input, making them a dynamical system. This feature of the RNNs make them attractive for processing serially correlated time series. Unlike the ANNs and traditional time series models, the RNNs are flexible enough, and avoids taking the size of sliding windows into account because they can decide what to store and what to ignore during the learning process.

In real-time data, training neural networks involves the optimisation of non-convex objective functions, which makes

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

the computation of the gradient descent challenging and in turn leads to a costly or even infeasible learning process. One solution is to randomly assign a subset of the networks' weights, so that the resulting optimisation task can be formulated as a linear least-squares problem. Numerous experimental results indicate that such randomised models can match, or even improve, adaptable ones, with a number of favourable benefits, such as, simplicity of implementation, faster learning with less intervention from human beings, possibility of leveraging over all linear regression and classification algorithms. Such improvements in standard RNNs design were proposed independently by Maass et al. [2002] under the name of Liquid State Machines and Jaeger [2001] under the name of Echo State Networks (ESN). Over time, these types of models became known as Reservoir Computing (RC). RC assumes that supervised adaptation of all inter-connection weights are not necessary, and only training a memoryless supervised readout from it is sufficient to obtain good results. Thus, it avoids the shortcomings of the gradient-descent training of RNNs. RC is based on the computational separation between a dynamic reservoir and a recurrence-free readout (see details in Section (6.3)).

1.3.2 Modelling changing environment

1.3.2.1 Multiple characteristic timescales

In general, the dynamics of RNN models evolve according to precise timing, while real-world time series are characterised by multiple seasonalities (or patterns). Substantial literature exists on modelling different temporal resolutions (multiple characteristic timescales), especially in presence of long-term relations, among which the Long Short Term Memory (LSTM) became a reference (see Hochreiter et al. [1997]). Several variants have been proposed to improve the simultaneous learning of long and short time relationships (see Cho et al. [2014]). Other methods developed based on the idea that temporal relationships are hierarchically structured, so that RNNs should be organised accordingly (see El Hihi et al. [1995]). It led some models to be implemented by stacking multiple recurrent layers (see Graves [2013], Chung et al. [2015]). In Reservoir Computing, characterised by fading memory preventing to model slow periodicities, solutions were proposed to slow down the dynamics of the reservoir by considering digital bandpass filters. The latter encourages the decoupling of the dynamics within a single reservoir by identifying particular frequency-domain characteristics. A large amount of filters are initialised with random cutoff frequencies to encode a wide range of timescales in the recurrent layer, providing the correct timings to capture the target dynamics. Alternatively, the filters can be manually tuned based on the information of the frequency spectrum of the desired response signal (see Siewert et al. [2007], Wyffels et al. [2008]).

An important development in machine learning was the use of ensemble models, which simultaneously combine different types of classifiers. The idea being that classifiers of different types and/or using different data exhibit distinct strengths and weaknesses. As a result, it has been assumed that combining multiple classifiers could improve prediction accuracy (see Ho et al. [1994]). Jaeger [2004] discussed the fact that a single reservoir could not train a multiple superimposed oscillator (MSO). We know from classic kernel-based signal representation (Fourier and wavelet transform) that an individual kernel can represent an almost unique portion of the target signal. Further, a single reservoir can reproduce the behaviour of a sine-wave generator (see Jaeger [2001]). We can therefore construct an ESN with multiple reservoirs, each of which having a different configuration, to generate multiple sine waves of different frequencies. The problem being to define the grouping of neurons into several separate reservoir in a non-redundant manner.
Wiestra et al. [2005] explained that all the neurons in the same reservoir are coupled, when the task requires the simultaneous existence of multiple decoupled internal states. One solution was to consider an evolutionary-based reservoir design method, called Evolino. Xu et al. [2007] proposed the decoupled echo state network (DESN) involving the use of lateral inhibition. It is a structure making use of multiple reservoirs competing with each other through inhibitory connections and combining the internal states of all the reservoirs to form the output signal. Bianchi et al. [2017] proposed a network made of several recurrent groups of neurons trained to separately adapt to each timescale of the input signal. The recurrent layer of the network is randomly generated, but the connections must form

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

a structure with K groups of bandpass neurons where each group $\mathcal{C}_k$ contains N neurons strongly connected to the other neurons in $\mathcal{C}_k$. Further, a small amount of connections between neurons belonging to different groups are also generated. The bandpass parameters are learned via backpropagation.

In most of these methods a large number of parameters must be learned in order to adapt the network to process the right time scales. It results in a long training time with the possibility of overfitting data. In general, these parameters are retrieved through an analysis in the frequency domain or by a priori knowledge of the input data.

1.3.2.2 Long range dependence

In general, memory or persistent internal states can not be modelled by fading memory systems (see Definition (10.2.1)). However, the latter can be enlarged through feedback from trained readouts. Maass et al. [2005a] [2005b] presented a computational theory characterising the gain in computational power achieved through feedback in dynamical systems with fading memory. As a result, anti-persistent systems using feedback can obtain universal computational capabilities similar to those of persistent systems. It leads to high-dimensional attractor-based models capable of capturing the characteristics of persistent signal response.

In the field of autonomous robot locomotion, stable pattern generation and robustness against perturbations is necessary. For instance, learning rhythmic stable motions in a controllable way is usually modelled with central patterns generators (CPG). In this framework, the output feedback can generate a suitable slow dynamics with high precision. As a result, RC models with output feedback were proposed to generate CPGs with interesting properties. In order to properly feedback the forecast and to modify appropriately the output weights if needed, one needs to implement an online algorithm. That is, when the reseroir is trained online, the model is adapted to recognising patterns and predicting chaotic time series (see Wyffels et al. [2009], Antonik et al. [2016]).

Reinhart et al. [2008] [2009] provided a new approach for training offline, or online, algorithms based on the transient/attractor concepts called Associative Reservoir Computing (ARC). As an example, an associative connection between two entities is bidirectional. The process denotes the recall of one entity from the other. Further, Reinhart [2011] introduced the concept of forward and inverse models in RC models. If the forward relation is many-to-one, the reverse relation becomes ambiguous, that is, one-to-many. Thus, there is a loss of information. One solution is to use additional information that is integrated over time. It can be modelled with an adaptive dynamical system to facilitate bidirectional and continuous association. The author pointed out the necessity of having a feedback in bidirectional association context. The main idea being to solve the ambiguous inverse problems by letting the model remain in an attractor network by teaching the reservoir states to stay stable. At every time step, the algorithm makes the entire network (input, reservoir and output) converge, thus eliminating transient effects. It is very important when one considers an output feedback matrix as it might add transient effect. Since feedback of erroneous outputs into the network can lead to error amplification, the concept of output feedback dynamics is formalised and an output feedback stability criterion is proposed.

1.3.2.3 The variable weight neural network

In conventional neural networks, connection weights are fixed. That is, the characteristics of the neural network is invariant (the invariant type neural network). The adaptive ability of neural networks to an unknown environment depends on its generalisation capability. However, there exists a limit of generalisation ability in invariant type neural networks. Yasuda et al. [2006] proposed the neural network with variable connection weights, which changes its connection weights and its characteristic according to the changing environment. Analysing an obstacle avoidance problem for an electric-powered wheelchair, they split the weights in two terms where the first term are basic quantities giving the basic motion of obstacle avoidance and the second terms adjust the connection weights in order to adapt the neural network to the change of environment in the vicinity of the wheelchair. The latter are calculated by using another neural network, whose inputs are outputs of PSD sensors. As a result, the law producing avoidance orders,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

changes according to the current condition of the obstacle in the vicinity of the wheelchair. Thus, this mechanism adapts the characteristics of neural network to the changing environment at every moment and corrects the generation algorithm of obstacle avoidance operation appropriately.
Lam et al. [2014] presented the variable weight neural network (VWNN), allowing its weights to be changed in operation according to the characteristic of the network inputs. Analysing the problem of surface material recognition and epilepsy seizure phases recognition, they demonstrated its ability to adapt to different characteristics of input data resulting in better performance than with fixed weights neural networks. A VWNN consists of two traditional neural networks, namely tuned and tuning neural networks. The former is the one which actually classifies the input data, while the latter provides the weights to the tuned neural network according to the characteristic of the input data. The VWNN works on the principle that the connection weights are function to the external input signal. Thus, in theory, the VWNN can be viewed as an infinite number of traditional neural networks with fixed weights.
A description is given in Appendix (6.4.1) and an application to financial time series is presented.

1.3.3 Interpreting neural networks

Neural networks (NNs) are based on predefined equations or formulae (see Section (10.1)) and have the capability of modelling complex statistical interactions between features for automatic feature learning. In order to perform a task, data scientists create an architecture (topology) whose (meta)-parameters are able (or not) to provide correct answers when some new input data is presented. These parameters are the key to their knowledge (see Palmer et al. [2002]), which depends on their generalisation ability.
Given this singular way of learning, NNs have been treated as black box models for their lack of interpretability compared with classical statistical models. That is, their inability to know in an explicit way the relations established between explanatory variables (input) and dependent variables (output).
Since the recent applications of NNs in the industry are intended to make critical decisions, it has become paramount to understand how these models make predictions. To do so, academics focused on explaining individual feature importance and determining the contribution of explanatory variables. One need first to discover (detect) the interactions and then interpret them, which is not an easy task. The methods used for detecting interactions from NNs are usually statistical methods. In general, they rely on the assumption that the response signal is stationary with independent and identically distributed increments. Further, these tests rely on the idea that NNs are capable of successfully modelling complex statistical interactions.

1.3.3.1 Explaining the input feature importance

Several approaches have been proposed to interpret NNs, classified as follows:

1. direct interpretation: explaining individual feature importance (Ross et al. [2017]), developing attention-based models, which illustrate where neural networks focus during inference, providing model-specific visualisations, such as feature map and gate activation visualisations (see Yosinski et al. [2015]).

2. indirect interpretation: post-hoc interpretation of feature importance (see Ribeiro et al. [2016]), knowledge distillation to simpler interpretable models.

The discovery of interactions (or statistical interaction detection) has become an essential part of the literature on neural networks. Two general approaches exist:

1. to conduct individual tests for each combination of features. Examples are ANOVA and Additive Groves (see Sorokina et al. [2008]). Both methods are computationally expensive in dimension higher than three.

2. to pre-specify all interaction forms of interest, then use Lasso to simultaneously select which are important. They are quick at selecting interactions but require specifying all interaction terms of interest (see Sun [1999]).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Interpreting NNs is far more difficult, but new methods have recently developed for traditional feedforward form and deep architectures. Methods such as feature map visualisation, de-convolution, saliency maps have been especially important to the vision community for understanding how convolutional networks represent images. In the case of long short-term memory networks (LSTMs), a research direction has studied multiplicative interactions in the unique gating equations of LSTMs to extract relationships between variables across a sequence (see Arras et al. [2017], Murdoch et al. [2018]). Tsang et al. [2018] proposed the Neural Interaction Detection (NID), which detects statistical interactions of any order or form captured by a feed-forward neural network, by examining its weight matrices. Top-K true interactions are determined from interaction rankings by using a special form of generalised additive model, which accounts for interactions of variable order.

1.3.3.2 Determining the contribution of explanatory variables

Several methods have been proposed to assess the relative importance (contribution) of each explanatory variable: methods from the family of sensitivity analyses (SA) (perturb method, profile method) (see Cortez et al. [2013]), numeric sensitivity analyses (NSA) computing the slope between input and output (partial derivative method (PaD)) and even the curvature effects (second order derivatives), and methods specific to the topology of NNs (connection weights method). In general, an optimal NN architecture is selected and several variable contribution methods are applied. However, when applying several methods to optimal or suboptimal NN architecture, the importance ranking of variables differs from method to method (high variability), indicating their inherent instability. As a result, there is no consensus on which model is the best for determining the contribution of variables. The difficulty of choosing a single optimal NN model led some authors to consider a set of good-performing NN models called neural network committee (NNC) (see Cao et al. [2008]). To mitigate these variabilities, De Ona et al. [2014] suggested to apply the methods on a set of NNs with the same architecture and trained with identical learning algorithm, activation functions, momentum value and learning ratio. The calculation of the average values of the relative importance of variables was used to determine the contribution of variables. The variables Frequency, Speed, Punctuality and Proximity are classified as the most important by all methods.

1.3.4 Optimisation for machine learning

1.3.4.1 Function estimation

In machine learning, given the samples (x_i, y_i), $i = 1, 2, ..., m$, of m pairs of input vector $x_i \in \mathbb{R}^{N_i}$ and labelled output vectors $y_i \in \mathbb{R}^{N_o}$, a task for supervised learning consists in learning a functional relation between the input x_i and the desired output y_i, for the training set (see Section (10.1)).
The learning can be understood as finding a mapping $h(\cdot, w)$ such that $h(x_i, w) \approx y_i$ with w an N_w dimensional vector of parameters to be learned. Thus, the goal of machine learning is to minimise the expected loss (also called risk function)

$$\widehat{L}(h) = E[L(h(X), Y)]$$

However, we do not know the joint probability distribution $P(X, Y)$ and can not estimate it. One solution is to consider Empirical Risk minimisation:

- we substitute the sample mean for the expectation
- minimise empirical loss:

$$\overline{L}(h) = \frac{1}{m} \sum_{i=1}^{m} L(h(x_i), y_i)$$

  Also known as (A.K.A.) sample average approximation (see Appendix (24.3)).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Mathematically, we want to find

$$w^* = \arg\min_w \overline{L}(w)$$

where $\overline{L}(w) = \frac{1}{m}\sum_{i=1}^m L(w; x_i, y_i))$ approximates the expected value of the loss. We can also write the loss function as $L(h(x_i, w), y_i)$.
In general, the relation between the input x_i and the desired output y_i can either be solved by a linear model as

$$\widehat{y}_i = W^\top x_i$$

where $W \in \mathbb{R}^{N_i \times N_o}$ is a weight matrix, or by a nonlinear model

$$\widehat{y}_i = f(W^\top x_i)$$

where $f(\cdot)$ is the activation function.
The training of neural networks (NNs) depends on the adaptation of free network parameters, the weight values. Thus, it is an optimisation task where the result is to find optimal weight set of the network that reduce an error function E. It can be formulated as the minimisation of an error function in the space of connection weights (see Chapter (18)). In the case of our loss function the minimisation problem is

$$\min_W \widehat{L}(h, W)$$

In practice, the function f in the above minimisation problem is not given explicitly but only implicitly through some examples. While convex problems are theoretically and algorithmically much more tractable, the complexity of non-convex problems can grow enormously (NP-hard). Global optimisation addresses the problems of non-convex optimisation.

1.3.4.2 Parameters estimation

The methods for training neural networks (NNs) are backpropagation (BP), Levenberg-Marquadt (LM), Quasi-Newton (QN). In general, the error backpropagation method (EBP), based on gradient method, is preferred. However, the objective function describing the neural networks training problem is a multi-modal function, so that the algorithms based on gradient methods can easily be stuck in local extremes. To avoid this problem one can consider using a global search optimisation (see Section (20.2)).

Over time, the relationship between mathematical programming models (MP) and machine learning models have been increasingly coupled. Methods developed for introducing constraints into the learning model in view of adding domain knowledge to enforce non-negativity and sparsity in dimensionality reduction methods. Further, methods were developed for solving existing machine learning models more efficiently. As data set size grows, MP models became inadequate and methods were developed to exploit the properties of learning problems in machine learning. In turn, machine learning has motivated advances in mathematical programming: the optimisation problems arising from large scale machine learning and data mining far exceed the size of the problem typically reported in the mathematical programming literature.

Generally, an ANN model is specified by its topology, node characteristics and the training algorithms. There are a variety of ANNs with various topologies, node properties and training algorithms. See for instance Section (6.3) in the case of Reservoir Computing. These characteristics (or features) must be considered when designing a neural network model, leading to large families of models. The data scientist must select an appropriate family of models and massages the data into a format amenable to modelling. Then the model is typically trained (Training) by solving a core optimisation problem that optimises the variables (or parameters) of the model with respect to the selected loss function and possibly some regularisation function. The efficiency of an optimisation method often depends on the choosing of a number of behavioural parameters (meta parameters). In that case, the mapping becomes

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$h(x_i, w, \theta) \approx y_i$$

with θ an N_θ dimensional vector of parameters to be learned. Hence, in the process of model selection and validation (), the core optimisation problem may be solved many times. It led researchers to combine mathematical programming with machine learning. The interaction of state-of-the-art machine learning and mathematical programming has been called Large Scale Optimization and Machine Learning leading to improved machine learning models as well as new uses of mathematical programming in machine learning.

One solution consists in performing an additional optimisation aiming at minimising the RMSE on these meta parameters Θ. It can be done on the validation sample. In that case, the minimisation problem is

$$\min_{\Theta} \hat{L}(h, W^*, \Theta)$$

Pedersen et al. [2008b] proposed an overlaying optimisation method known as meta-optimisation. The idea is to have an optimisation method act as an overlaying meta-optimiser, trying to find the best performing behavioural parameters for another optimisation method (see Pedersen et al. [2008a]).

1.4 Computer vision

See textbooks by Davies [2005], Szeliski [2010], Parker [2011], among others.

1.4.1 Description

1.4.1.1 An overview

In the 1960s, computer vision in artificial intelligence (AI) was meant to mimic the human visual system, as a stepping stone to endowing robots with intelligent behaviour (see Boden [2008]). Since some areas of AI deal with autonomous planning or deliberation for robotical systems to navigate through an environment, a detailed understanding of these environments was required to navigate through them. Information about the environment could be provided by a computer vision system, acting as a vision sensor and providing high-level information about the environment and the robot.
In the 1980s, more studies were produced on rigorous mathematical analysis and quantitative aspects of computer vision which could be treated within the same optimisation framework as regularisation and Markov random fields. The 1990s marked the first time statistical learning techniques were used in practice to recognise faces in images. Toward the end of the 1990s, a significant change came about with the increased interaction between the fields of computer graphics and computer vision. This included image-based rendering, image morphing, view interpolation, panoramic image stitching and early light-field rendering. Recent work has seen the resurgence of feature-based methods, used in conjunction with machine learning techniques and complex optimisation frameworks.

Nowadays, computer vision is an interdisciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do. Computer vision tasks include methods for acquiring, processing, analysing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information. Here, understanding means the transformation of visual images (the input of the retina) into descriptions of the world that can interface with other thought processes and elicit appropriate action. It can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory. It involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding. As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models for the construction of computer vision systems.

1.4.1.2 Some tasks

All applications in computer vision employ a range of computer vision tasks. Some of them are as follows:

- Recognition: The classical problem in computer vision, image processing, and machine vision is that of determining whether or not the image data contains some specific object, feature, or activity. Some varieties of the recognition problem are as follows:
  - Object recognition (or classification): one or several pre-specified or learned objects, or object classes, can be recognised, usually together with their 2D positions in the image, or 3D poses in the scene. Blippar, Google Goggles and LikeThat provide stand-alone programs that illustrate this functionality.
  - Identification: an individual instance of an object is recognised. Examples include identification of a specific person's face or fingerprint, identification of handwritten digits, or identification of a specific vehicle.
  - Detection: the image data are scanned for a specific condition. Examples include detection of possible abnormal cells or tissues in medical images or detection of a vehicle in an automatic road toll system. Detection based on relatively simple and fast computations is sometimes used for finding smaller regions of interesting image data which can be further analysed by more computationally demanding techniques to produce a correct interpretation.

  Nowadays, the best algorithms for such tasks are based on convolutional neural networks (see Section (9.1.1)). An illustration of their capabilities is given by the ImageNet Large Scale Visual Recognition Challenge; this is a benchmark in object classification and detection, with millions of images and hundreds of object classes. Performance of convolutional neural networks, on the ImageNet tests, is now close to that of humans (see Russakovsky et al. [2014]).
  Some specialised tasks based on recognition are:

  - Content-based image retrieval: finding all images in a larger set of images which have a specific content. The content can be specified in different ways, for example in terms of similarity relative a target image (give me all images similar to image X), or in terms of high-level search criteria given as text input (give me all images which contains many houses, are taken during winter, and have no cars in them).
  - Pose estimation: estimating the position or orientation of a specific object relative to the camera. An example application for this technique would be assisting a robot arm in retrieving objects from a conveyor belt in an assembly line situation or picking parts from a bin.
  - Optical character recognition (OCR): identifying characters in images of printed or handwritten text, usually with a view of encoding the text in a format more amenable to editing or indexing. It is a common method of digitising printed texts so that they can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes such as cognitive computing, machine translation, (extracted) text-to-speech, key data and text mining.
  - 2D code reading: reading of 2D codes such as data matrix and QR codes.
  - Facial recognition: Face recognition is a PC innovation that decides the positions and sizes of human faces in self-assertive digital pictures. It distinguishes facial components and overlooks whatever, for example, structures, trees and bodies.
  - Shape Recognition Technology (SRT) in people counter systems differentiating human beings (head and shoulder patterns) from objects.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Motion analysis: Several tasks relate to motion estimation where an image sequence is processed to produce an estimate of the velocity either at each points in the image or in the 3D scene, or even of the camera that produces the images. Some of these tasks are as follows:
  - Egomotion: determining the 3D rigid motion (rotation and translation) of the camera from an image sequence produced by the camera.
  - Tracking: following the movements of a (usually) smaller set of interest points or objects (such as vehicles, humans or other organisms) in the image sequence.
  - Optical flow: to determine, for each point in the image, how that point is moving relative to the image plane, that is, its apparent motion. This motion is a result both of how the corresponding 3D point is moving in the scene and how the camera is moving relative to the scene.
- Scene reconstruction: Given one or more images of a scene, or a video, it aims at computing a 3D model of the scene. In the simplest case the model can be a set of 3D points. More sophisticated methods produce a complete 3D surface model. The advent of 3D imaging not requiring motion or scanning, and related processing algorithms is enabling rapid advances in this field. Grid-based 3D sensing can be used to acquire 3D images from multiple angles. Algorithms are now available to stitch multiple 3D images together into point clouds and 3D models (see Soltani et al. [2017]).
- Image restoration: The main goal is the removal of noise (sensor noise, motion blur) from images. The simplest possible approach for noise removal is various types of filters such as low-pass filters or median filters. More sophisticated methods assume a model of how the local image structures look, to distinguish them from noise. By first analysing the image data in terms of the local image structures, such as lines or edges, and then controlling the filtering based on local information from the analysis step, a better level of noise removal is usually obtained compared to the simpler approaches.

1.4.1.3 Computer vision systems

The organisation of a computer vision system is highly application-dependent. Some systems are stand-alone applications that solve a specific measurement or detection problem, while others constitute a sub-system of a larger design which, for example, also contains sub-systems for control of mechanical actuators, planning, information databases, man-machine interfaces (see Davies [2005]). The specific implementation of a computer vision system also depends on whether its functionality is pre-specified or if some part of it can be learned or modified during operation. Some typical functions in many computer vision systems are as follows:

- Image acquisition: A digital image is produced by one or several image sensors, which, besides various types of light-sensitive cameras, include range sensors, tomography devices, radar, ultra-sonic cameras. Depending on the type of sensor, the resulting image data is an ordinary 2D image, a 3D volume, or an image sequence. The pixel values typically correspond to light intensity in one or several spectral bands (gray images or colour images), but can also be related to various physical measures, such as depth, absorption or reflectance of sonic or electromagnetic waves, or nuclear magnetic resonance.
- Pre-processing: Before a computer vision method can be applied to image data in order to extract some specific piece of information, it is usually necessary to process the data in order to assure that it satisfies certain assumptions implied by the method. Examples are:
  - Re-sampling to assure that the image coordinate system is correct.
  - Noise reduction to assure that sensor noise does not introduce false information.
  - Contrast enhancement to assure that relevant information can be detected.
  - Scale space representation to enhance image structures at locally appropriate scales.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Feature extraction: Image features at various levels of complexity are extracted from the image data. Typical examples of such features are:
  - Lines, edges and ridges.
  - Localised interest points such as corners, blobs or points.

  More complex features may be related to texture, shape or motion.
- Detection/segmentation: At some point in the image processing, a decision is made about which image points or regions of the image are relevant for further processing. Some examples are as follows:
  - Selection of a specific set of interest points.
  - Segmentation of one or multiple image regions that contain a specific object of interest.
  - Segmentation of image into nested scene architecture comprising foreground, object groups, single objects or salient object parts (also referred to as spatial-taxon scene hierarchy), while the visual salience is often implemented as spatial and temporal attention.
  - Segmentation or co-segmentation of one or multiple videos into a series of per-frame foreground masks, while maintaining its temporal semantic continuity.
- High-level processing: At this step the input is typically a small set of data, for example a set of points or an image region which is assumed to contain a specific object. The remaining processing deals with, for example:
  - Verification that the data satisfy model-based and application-specific assumptions.
  - Estimation of application-specific parameters, such as object pose or object size.
  - Image recognition: classifying a detected object into different categories.
  - Image registration: comparing and combining two different views of the same object.
- Decision making: Making the final decision required for the application, for example:
  - Pass/fail on automatic inspection applications.
  - Match/no-match in recognition applications.
  - Flag for further human review in medical, military, security and recognition applications.

1.4.2 Digital images

1.4.2.1 Description

A pixel (pel or picture element) is a physical point in a raster image, or the smallest addressable element in an all points addressable display device. Thus, it is the smallest controllable element of a picture represented on the screen. A set of pixels is a 2d grid of square cells which have red (r), green (g), blue (b) colour channels and an opacity value. Each pixel is actually a very small square that is assigned a colour, and then arranged in a pattern to form the image.
A digital image is a numeric representation, normally binary, of a two-dimensional image. Digital images can usually be divided into two distinct categories. They are either bitmap (raster) files or vector graphics. In general

- Digital pictures and scanned images are bitmap files. These are sometimes also called raster images.
- Drawings made in applications like Adobe Illustrator or Corel Draw are saved as vector graphics. Vector graphics are more often used for abstract images such as logos.

Digital imaging or digital image acquisition is the creation of a digitally encoded representation of the visual characteristics of an object, such as a physical scene or the interior structure of an object. The term is often assumed to imply or include the processing, compression, storage, printing, and display of such images.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.4.2.2 The categories

1.4.2.2.1 Bitmap Bitmap (or raster) images are a collection of bits that form an image. The image consists of a matrix of individual dots (or pixels) that all have their own colour (described using bits, the smallest possible units of information for a computer). Thus, a bitmap is just pixels with (r), (g), (b) values at each pixel. When you zoom in on a bitmap image you can see the individual pixels that make up that image. Bitmap graphics can be edited by erasing or changing the colour of individual pixels using a program such as Adobe Photoshop.
Since bitmaps are resolution dependent, it is impossible to increase or decrease their size without sacrificing a degree of image quality. When you increase the size of a bitmap image, the software has to create new pixels. When creating pixels, the software has to estimate the colour values of the new pixels based on the surrounding pixels. This process is called interpolation.

1.4.2.2.2 Vector graphics Vector graphics are computer graphics images that are defined in terms of 2D points, which are connected by lines and curves to form polygons and other shapes. Each of these points has a definite position on the x- and y-axis of the work plane and determines the direction of the path; further, each path may have various properties including values for stroke colour, shape, curve, thickness, and fill.
Vector drawings can usually be scaled without any loss in quality. This makes them ideal for company logos, maps or other objects that have to be resized frequently. Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation. The SVG specification is an open standard developed by the World Wide Web Consortium (W3C) since 1999. SVG images and their behaviours are defined in XML text files.
Not all vector drawings can be scaled as much as you like:

- Drawings containing trapping information can only be scaled up to 20 percent larger or smaller.
- Thin lines may disappear if a vector drawing is reduced too much.
- Small errors in a drawing may become visible as soon as it is enlarged too much.

It is fairly easy to create a vector-based drawing that is very difficult to output. Especially the use of tiles (small objects that are repeated dozens or hundreds of times) and Corel Draw lens effects can lead to very complex files.

1.4.2.3 Format

Optical Character Recognition (OCR) (see Section (1.5.3.1)) expects a bitmap as input. The Image-base is a bitmap, so it is already in the format expected by OCR (the resolution and colour representation may need to be altered as a preconditioning step of the OCR with a standard tool). Bitmap data can take up a lot of room. Unlike bitmaps, vector images are not based on pixel patterns, but instead use mathematical formulas to draw lines and curves that can be combined to create an image from geometric objects such as circles and polygons. Vector images have some important advantages over bitmap images. They tend to be smaller than bitmap images. This is because a bitmap image has to store colour information for each individual pixel that forms the image. A vector image just has to store the mathematical formulas that make up the image, which take up less space. Vector images are also more scalable than bitmap images.

Bitmap data can be saved in a wide variety of file formats. The three most popular image formats used on the Web (PNG, JPEG, and GIF) are bitmap formats. The Vector-base image, such as SVG, is in XML format and would need to be first converted into a bitmap before being processed as above. Bitmap formats are best for images that need to have a wide range of colour gradations, such as most photographs. Vector formats, on the other hand, are better for images that consist of a few areas of solid colour. Examples of images that are well suited for the vector format include logos and type.

It is sometimes necessary to transform images from bitmap data to vector data or back. Some possible uses include:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- If we scan or photograph a logo, it is a bitmap image. If it is going to be used often in a layout it is more practical to have that logo as a vector drawing. That reduces its file size and you can change the image size without worrying about any loss in quality.
- Vector drawings often have to be converted to bitmaps if they will be used on a web page.
- Vector drawings are sometimes too complicated for a RIP to be output on film or plate. Converting them to a bitmap simplifies the file.

Vector images can easily be converted to bitmaps. This process is called rasterising.
One can convert images from one mode to the other:

- From bitmap data to vector graphics: the process to convert a bitmap image to vector data is called Outlining or Vectorising. Some drawing applications such as Adobe Illustrator and Corel Draw have this option built in. There are also separate programs available to vectorise bitmap images.
- From vector graphics to bitmap data:
  - Many drawing applications can store vector data as bitmap files as well.
  - You can always view a vector file on screen, then take a screen capture and save this screen capture as a bitmap image.
  - Photoshop can open some vector file formats and rasterise the file so that it becomes a bitmap file. A pop-up menu allows you to define the resolution and colour mode of the bitmap data.

1.4.3 Visual detection

1.4.3.1 Overview

Visual perception is the ability to interpret the surrounding environment using light in the visible spectrum reflected by the objects in the environment. The resulting perception is also known as vision. The various physiological components involved in vision are referred to collectively as the visual system. Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. Hence, visual detection is concerned with identifying an object in an image. It is widely used in computer vision tasks such as image annotation, activity recognition, face detection, face recognition, video object co-segmentation. It is also used in tracking objects, for example tracking a ball during a football match, tracking movement of a cricket bat, or tracking a person in a video.

Every object class has its own special features that helps in classifying the class. For instance, all circles are round. Thus, object class detection uses these special features. For example, when looking for circles, objects that are at a particular distance from a point (i.e. the center) are sought. Similarly, when looking for squares, objects that are perpendicular at corners and have equal side lengths are needed. A similar approach is used for face identification where eyes, nose, and lips can be found and features like skin color and distance between eyes can be found.

1.4.3.2 Description

Visual perception involves mapping pixel values and light information onto a model of the universe to infer the surroundings. General scene understanding requires complex visual tasks such as segmenting a scene into component parts, recognising what those parts are, and disambiguating between visually similar objects.
Most methods for object detection generally fall into either neural network-based or non-neural approaches:

1. Non-neural approaches: It is a two-step process. In the first step we define features

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- item Scale-invariant feature transform (SIFT): It is a feature detection algorithm in computer vision to detect and describe local features in images.
- Histogram of oriented gradients (HOG) features: The technique counts occurrences of gradient orientation in localised portions of an image.

Then, using a classifier such as support vector machine (SVM), we do the classification. The most common approach to grasp detection is a sliding window detection framework which uses a classifier to determine whether small patches of an image constitute good grasps for an object in that image. This type of system requires applying the classifier to numerous places on the image. Patches that score highly are considered good potential grasps.

2. Neural network approaches: They are able to do end-to-end object detection without specifically defining features, and are typically based on convolutional neural networks (CNN). They apply a single network once to an image and predict grasp coordinates directly. Even though the network is comparatively large since it is only applied once to an image, it massively increases performance. Rather than looking only at local patches the network uses global information in the image to inform its grasp predictions, making it significantly more accurate. Some of these models are:

   - Region Proposals: R-CNN (see Girschick [2014]), Fast R-CNN (see Girschick [2015]), Faster R-CNN (see Shaoqing [2015]), cascade R-CNN (see Pan et al. [2019])
   - Single Shot MultiBox Detector (SSD)
   - You Only Look Once (YOLO) (see Redmon et al. [2015], [2018])
   - Retina-Net by Facebook AI Research (FAIR) (see Lin [2020])
   - Deformable convolutional networks

Note, object detection models can also be classified into

1. Single-stage detectors: A very large set of candidate object locations is regularly sampled across an image (100k locations), which densely cover spatial positions, scales and aspect ratios.
2. Two-stage detectors: In the first stage, region proposal network (RPN) narrows down the number of candidate object location to a small number (such as $1 - 2$k), filtering out most background samples. In the second stage, classification is performed for each candidate object location. Two-stage detectors are often more accurate but at the cost of being slower. Example include the Faster R-CNN.

Nowadays, the most advanced algorithm can work with a resolution of 4096×4096, but the training is very expensive.

1.4.3.3 Some detection system

As discussed above, most detection systems repurpose classifiers or localisers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections.

1.4.3.3.1 YOLO detection system You only look once (YOLO) is a state-of-the-art, real-time object detection system. It works to label several areas inside an image. YOLO is a single-pass object detection. The model is similar to LSTM (see Section (6.2)), but it works on images. YOLO applies a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities (see Redmon et al. [2015]). As YOLO outputs a series of rectangle boundaries showing the positions of key areas, the accuracy is often not too high, since it usually detects single of no more than 10 objects. The detection of multiple objects like a swarm performs very badly.
The model has several advantages over classifier-based systems. It looks at the whole image at test time, so its

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than $1000x$ faster than R-CNN and 100x faster than Fast R-CNN. See our paper for more details on the full system. Darknet prints out the objects it detected, its confidence, and how long it took to find them. Redmon et al. [2018] made some updates to YOLO.

1.4.3.3.2 RetinaNet detection system It has been found that there is extreme foreground-background class imbalance problem in single-stage detectors making their performance inferior to that of two-stage detectors. RetinaNet (see Lin [2020]) is a popular single-stage detector, which is accurate and runs fast. It uses a feature pyramid network to efficiently detect objects at multiple scales and introduces a new loss, the Focal loss function (see Section (10.1.2.2)), to alleviate the problem of the extreme foreground-background class imbalance. RetinaNet is formed of ResNet plus FPN, which act as backbone for feature extraction, plus two task-specific subnetworks for classification and bounding box regression. It achieves state-of-the-art performance, outperforms Faster R-CNN, the well-known two-stage detectors.
The sample label data is made up of the input images, bounding boxes for the objects present and their class ids. These raw labels need to be transformed into targets for training. This transformation consists of the following steps:

- Generating anchor boxes for the given image dimensions
- Assigning ground truth boxes to the anchor boxes
- The anchor boxes that are not assigned any objects, are either assigned the background class or ignored depending on the IOU
- Generating the classification and regression targets using anchor boxes

1.4.4 Parsing documents

1.4.4.1 Terminology

1.4.4.1.1 XML Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The W3C's XML 1.0 Specification and several other related specifications, all of them free open standards, define XML.
XML has been used to describe structured data and to serialise objects. XML can contain mixed content, that is, additional content between element tags. XML separates data from metadata (via the use of elements and attributes). XML also has the concept of schema. This permits strong typing, user-defined types, predefined tags, and formal structure, allowing for formal validation of an XML stream in a portable way.
XML and its extensions have regularly been criticised for verbosity, complexity and redundancy. Mapping the basic tree model of XML to type systems of programming languages or databases can be difficult, especially when XML is used for exchanging highly structured data between applications, which was not its primary design goal. Other criticisms attempt to refute the claim that XML is a self-describing language. JSON, YAML, and S-Expressions are frequently proposed as simpler alternatives that focus on representing highly structured data rather than documents, which may contain both highly structured and relatively unstructured content.

1.4.4.1.2 JSON JavaScript Object Notation (JSON) is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute: value pairs and array data types (or any other serialisable value). It is a very common data format used for asynchronous browser-server communication, including as a replacement for XML in some AJAX-style systems.
JSON is promoted as a low-overhead alternative to XML as both of these formats have widespread support for creation, reading, and decoding in the real-world situations where they are commonly used.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.4.4.1.3 Parsing Parsing, syntax analysis, or syntactic analysis is the process of analysing a string of symbols, either in natural language, computer languages or data structures, conforming to the rules of a formal grammar. Within computational linguistics the term is used to refer to the formal analysis by a computer of a sentence or other string of words into its constituents, resulting in a parse tree showing their syntactic relation to each other, which may also contain semantic and other information. Traditional sentence parsing is often performed as a method of understanding the exact meaning of a sentence or word, sometimes with the aid of devices such as sentence diagrams. It usually emphasises the importance of grammatical divisions such as subject and predicate.
Within computer science, the term is used in the analysis of computer languages, referring to the syntactic analysis of the input code into its component parts in order to facilitate the writing of compilers and interpreters. The term may also be used to describe a split or separation.
A parser is a software component that takes input data (frequently text) and builds a data structure, often some kind of parse tree, abstract syntax tree or other hierarchical structure, giving a structural representation of the input while checking for correct syntax. The parsing may be preceded or followed by other steps, or these may be combined into a single step.
In the case of data languages, a parser is often found as the file reading facility of a program, such as reading in HTML or XML text. These examples are markup languages.

1.4.4.2 OCR

As discussed above, OCR is the automated conversion of images of typed, handwritten, or printed text, into machine encoded text. OCR is generally an offline process, which analyses a static document. Some cloud based services provide an online OCR API service. The techniques for OCR are as follows:

- Pre-processing: This is to improve the chances of successful recognition. It includes
  - De-skew: If the document is not aligned properly when scanned, it may need to be tilted a few degrees clockwise or counterclockwise in order to make lines of text perfectly horizontal or vertical.
  - Despeckle: Remove positive and negative spots, smoothing edges.
  - Binarisation: Convert an image from color or greyscale to black-and-white (called a binary image because there are two colors). It is performed as a simple way of separating the text (or any other desired image component) from the background. In addition, the effectiveness of the binarisation step influences to a significant extent the quality of the character recognition stage and the careful decisions are made in the choice of the binarisation employed for a given input image type.
  - Line removal: Cleans up non-glyph boxes and lines.
  - Layout analysis (or zoning): Identifies columns, paragraphs, captions, etc. as distinct blocks. Especially important in multi-column layouts and tables.
  - Line and word detection: Establishes baseline for word and character shapes, separates words if necessary.
  - Script recognition: In multilingual documents, the script may change at the level of the words and hence, identification of the script is necessary, before the right OCR can be invoked to handle the specific script.
  - Character isolation (or segmentation): For per-character OCR, multiple characters that are connected due to image artifacts must be separated; single characters that are broken into multiple pieces due to artifacts must be connected.
  - Normalise aspect ratio and scale.
- Text recognition: There are two basic types of core OCR algorithm, which may produce a ranked list of candidate characters.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1. Matrix matching involves comparing an image to a stored glyph on a pixel-by-pixel basis; it is also known as pattern matching, pattern recognition, or image correlation. It relies on the input glyph being correctly isolated from the rest of the image, and on the stored glyph being in a similar font and at the same scale. This technique works best with typewritten text and does not work well when new fonts are encountered.
2. Feature extraction decomposes glyphs into features like lines, closed loops, line direction, and line intersections. The extraction features reduces the dimensionality of the representation and makes the recognition process computationally efficient. These features are compared with an abstract vector-like representation of a character, which might reduce to one or more glyph prototypes. General techniques of feature detection in computer vision are applicable to this type of OCR, which is commonly seen in intelligent handwriting recognition and most modern OCR software. Nearest neighbour classifiers such as the K-nearest neighbours algorithm are used to compare image features with stored glyph features and choose the nearest match.

Some software, such as Cuneiform and Tesseract, use a two-pass approach to character recognition. The second pass is known as adaptive recognition and uses the letter shapes recognized with high confidence on the first pass to recognize better the remaining letters on the second pass. This is advantageous for unusual fonts or low-quality scans where the font is distorted (e.g. blurred or faded). Modern OCR software such as OCRopus, or Tesseract, uses neural networks which were trained to recognise whole lines of text instead of focusing on single characters.

- Post-processing: OCR accuracy can be increased if the output is constrained by a lexicon, that is, a list of words that are allowed to occur in a document. For example, all the words in the English language, or a more technical lexicon for a specific field. However, this technique can be problematic if the document contains words not in the lexicon, like proper nouns. Tesseract uses its dictionary to influence the character segmentation step, for improved accuracy.
  The output stream may be a plain text stream or file of characters, but more sophisticated OCR systems can preserve the original layout of the page and produce, for example, an annotated PDF that includes both the original image of the page and a searchable textual representation. Knowledge of the grammar of the language being scanned can also help determine if a word is likely to be a verb or a noun, for example, allowing greater accuracy.

- Application specific optimisation: Some major OCR technology providers tweak OCR systems to deal more efficiently with specific types of input. Beyond an application-specific lexicon, better performance can be obtained by taking into account business rules, standard expression, or rich information contained in color images. This strategy is called Application-Oriented OCR, or Customized OCR, and has been applied to OCR of license plates, invoices, screenshots, ID cards, driver licenses, and automobile manufacturing.

1.4.4.3 PDF

The Portable Document Format (PDF) is a file format developed by Adobe in the 1990s to present documents, including text formatting and images, in a manner independent of application software, hardware, and operating systems. Based on the PostScript language, each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts, vector graphics, raster images and other information needed to display it.

1.4.4.3.1 Content A PDF file is often a combination of vector graphics, text, and bitmap graphics. The basic types of content in a PDF are:

- Text stored as content streams (not encoded in plain text)
- Vector graphics for illustrations and designs that consist of shapes and lines

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Raster graphics (bitmap) for photographs and other types of image
- Multimedia objects in the document

In later PDF revisions, a PDF document can also support links (inside document or web page), forms, JavaScript, or any other types of embedded contents that can be handled using plug-ins. The PDF combines three technologies:

1. A subset of the PostScript page description programming language, for generating the layout and graphics.
2. A font-embedding/replacement system to allow fonts to travel with the documents.
3. A structured storage system to bundle these elements and any associated content into a single file, with data compression where appropriate.

A PDF file is a 7-bit ASCII file, except for certain elements that may have binary content.

1.4.4.3.2 Imaging model PDF graphics use a device-independent Cartesian coordinate system to describe the surface of a page. A PDF page description can use a matrix to scale, rotate, or skew graphical elements. A key concept in PDF is that of the graphics state, which is a collection of graphical parameters that may be changed, saved, and restored by a page description.
Vector graphics in PDF are constructed with paths. Paths are usually composed of lines and cubic Bezier curves, but can also be constructed from the outlines of text.

1.4.4.3.3 Text Text in PDF is represented by text elements in page content streams. A text element specifies that characters should be drawn at certain positions. The characters are specified using the encoding of a selected font resource.

1.4.4.3.4 Fonts A font object in PDF is a description of a digital typeface. It may either describe the characteristics of a typeface, or it may include an embedded font file. The latter case is called an embedded font while the former is called an unembedded font. The font files that may be embedded are based on widely used standard digital font formats.

1.4.4.3.5 Conversion Portable Document Format (PDF) is a file format in use to present and exchange documents reliably. PDF are not editable, but with PDF Conversion, we can convert a PDF to a .txt, .doc, .xlsx etc. files.

1.4.4.4 PDF parser

1.4.4.4.1 Description Pdf-parser is a command-line program that parses and analyses PDF documents. It provides features to extract raw data from PDF documents, like compressed images. A PDF Parser (also called PDF scraper) is a software which can be used to extract data from PDF documents. It is a PDF data extraction solution. PDF parsing solutions are typically used to extract:

- Text paragraphs
- Single data fields (dates, tracking numbers, ...)
- Tabular data (tables and lists)
- Images

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

While scanned images are easily readable for humans, computers are not capable of understanding the scanned image text without first applying a method called Optical Character Recognition (OCR) (see Section (1.4.4.2)). Once some documents went through an OCR PDF Scanner and actually contain text data and not just images, it is possible to manually copy and paste parts of the text. However, this method is tedious, error-prone and not scalable. Even when we want to extract table data, selecting the table with our mousepointer and pasting the data into Excel will give us decent results in a lot of cases. We can also use a free tool called Tabula to extract table data from PDF files. Tabula will return a spreadsheet file which we probably need to post-process manually. Note, Tabula does not include an OCR engines.

1.4.4.4.2 Fully automated PDF data extraction software Automated PDF data extraction solutions come in different flavours, ranging from simple OCR tools to enterprise ready document processing and workflow automation platforms. Most systems share however a similar workflow:

1. Assemble batches of samples documents which acts as training data.
2. Train the system for each type of document we want to process.
3. Set up a process to automatically fetch documents, process them and dispatch the data.

Most advanced solutions use a combination of different techniques to train the data extraction system. A simple method is for example Zonal OCR where the user simply defines specific locations inside the document with a point and click system. More advanced techniques are based on regular expressions and pattern recognition.
After the initial training period, document data extraction systems offer a fast, reliable and secure solution to automatically convert PDF documents into structured data. Especially when dealing with many documents of the same type (Invoices, Purchase Orders, Shipping Notes, ...), using a PDF Parser is a viable solution.

1.4.4.4.3 Repetitive information It applies to documents of the same type such as invoices, purchase orders, reports. For instance, when clients send us multiple invoices, or we receive the same PDF template from a partner every month but the data within those PDFs will change, we (as a human) can easily extract information. PDF are designed for human extraction. This can be quantified (see Babelway). To do so, we look for:

- Fixed extraction: The value we want to extract is always present at a specific fixed position.
- Relative to label/image: The value we want to extract is relative to a label or image.
- Within a table or list of items: The values we want to extract are repeating themselves.

1.4.4.4.4 Abbyy FineReader FineReader is an all-in-one OCR and PDF software application for increasing business productivity when working with documents. It provides powerful, yet easy-to-use tools to access and modify information locked in paper-based documents and PDF files.
Convert PDF documents to searchable and editable formats. Digitising processes: Accurately convert scanned paper documents, images and PDFs to Word, Excel, searchable PDF and other formats. This is done by preserving document layout and structure (including headers, footnotes, page numbering, table structures, and more).

OCR Editor provides all the tools necessary for advanced document conversion, giving full control and enabling the best results, even for very complex documents. In the case where documents contain terminology from specialised fields, or use rare or decorative fonts, one can train FineReader to recognise them correctly and get even more accurate results.

1.4.4.5 Extracting tables

We want convert PDF documents containing tables into another format such as JSON.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.4.4.5.1 Towards a solution As discussed above, PDF is a structured binary format for storing semantically rich content consisting of text, vectors, images, 3D objects, forms, annotations and more. On the other hand, JSON is just a serialisation of a Javascript object. That is, PDF is a file format, whereas JSON is a data exchange format. The problem becomes that of mapping these two concepts to each other.
The content of PDF is not written in structured format unless it is tagged. Thus, the problem is basically about content extraction and formatting the result into JSON. Success will depend mainly on the type of content we want to extract from the PDF: text, images, metadata? We cannot convert all content and must customise the output only with relevant information. So, we first need to parse the PDF and then output the relevant information in whatever format we want. We can use some free PDF parsing libraries like PDFBox, Podofo, PDFtk and programmatically generate JSON files from the text, images or other content. For instance, one can save a PDF as XML with Adobe Acrobat. JSON is not an option, but once we have the XML file there are several XML to JSON converters out there.
There are essentially two different problems to solve:

1. Extract text data from the PDF files. Check if the PDF files are actually containing text data or if they consist of scanned images. If no text is returned, pipe the files through an OCR library. Use an OCR system to convert images into searchable PDF files. Such PDF files contain the scanned images as well as hidden text returned by the OCR.

2. Convert the extracted text into individual data fields (Title, Headline, Text, Date, Reference Number, ...) which are used to build the JSON data object. Converting unstructured or semi-structured text into a JSON object can be easy, challenging or plain impossible. It really depends on the kind of data we are dealing with and how granular the output needs to be.

Several software exist that convert PDF onto Json: pdf2json, PDF Multitool, Docparser. Docparser is a software solution specialised in transforming semi-structured documents (invoices, purchase orders, reports, ...) into structured data such as JSON, CSV, XML.

1.4.4.5.2 The difficulty of getting a solution As previously discussed, PDF parsing is a widespread problem. However, when it comes to complicated formatting like tables, all of the OCR systems are far from perfect. A lot of data scientists tackled this problem, but there is no well-established solution.
There are various approaches, and some of them use XML step at some point. Since PDF is a graphic (not semantic) format, there is no guarantee that the data is in any order. For some documents, the approaches described above can work like a charm, for others some text will be missed (some parts may be graphic symbols, for letters), the order can be different, and almost for sure it will not work for tables.
Even if some PDFs are good with existing software for pre-processing, it is an empirical question to know if it works for some selected problems. Due to the irregularity of the problem (a human-readable table graphics can be in many formats, some of those are NOT 2d arrays for a computer) it is hard to estimate how much can be done. Thus, this approach for PDF paring is not a general solution.

1.5 Natural language processing

Natural language processing (NLP) is a subfield of computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyse large amounts of natural language data.
See book by Lawler et al [1998], Jurafsky et al. [2009], Kurdi [2016] [2017], among others.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.5.1 An overview

1.5.1.1 Natural languages

A language is a structured system of communication used by humans consisting of spoken sounds in spoken languages or written elements in written languages. It can be defined as a system of signs governed by grammatical rules of combination to communicate meaning. For instance, all languages rely on the process of semiosis to relate signs to particular meanings. This definition stresses that human languages can be described as closed structural systems consisting of rules that relate particular signs to particular meanings. For example, oral, manual and tactile languages contain a phonological system that governs how symbols are used to form sequences known as words or morphemes, and a syntactic system that governs how words and morphemes are combined to form phrases and utterances. Thus, even though natural languages are spoken or signed, any language can be encoded into secondary media using auditory, visual, or tactile stimuli, including in writing, whistling, signing, or braille. This is because human language is modality-independent. Languages have the properties of productivity and displacement, and relies on social convention and learning. As an object of linguistic study, languages have two primary meanings: an abstract concept, and a specific linguistic system. A language consists of three parts:

1. Signs: A sign is anything that communicates a meaning, that is not the sign itself, to the interpreter of the sign. They can be composed of sounds, gestures, letters, or symbols. **Semiosis**, or sign process, is any form of activity, conduct, or process that involves signs, including the production of meaning. A sign is encoded and transmitted by a sender through a channel to a receiver who decodes it. The duality of the linguistic system means that linguistic structures are built by combining elements into larger structures that can be seen as layered.

2. Meanings: The meaning that is connected to individual signs, morphemes, words, phrases, and texts is called **semantics**. The arbitrariness of the linguistic sign means that there is no predictable connection between a linguistic sign and its meaning. That is, the convention is established socially and historically. Thus, languages must have a vocabulary of signs related to specific meaning. The array of arbitrary signs connected to specific meanings is called the lexicon, and a single sign connected to a meaning is called a lexeme. There are two notions to the predicate: (1) it completes an idea about the subject, such as what it does or what it is like, (2) it corresponds to the main verb (and potentially to any auxiliary verbs that accompany the main verb), whereas the arguments of that predicate (the subject and object noun phrases) are outside the predicate. All languages contain the semantic structure of predication, that is, a structure that predicates a property, state, or action. Semantics is the study of how speakers and interpreters assign truth values to statements, so that meaning is understood to be the process by which a predicate can be said to be true or false about an entity [5]. It has been complemented with more dynamic models of meaning that incorporate shared knowledge about the context in which a sign is interpreted into the production of meaning. Such models of meaning are explored in the field of pragmatics.

3. A code connecting signs with their meanings: The rules by which signs can be combined to form words and phrases are called syntax or **grammar**. It is the study of how meaningful elements called morphemes within a language can be combined into utterances. Morphemes can either be free to be moved around (they are called words), or bound to other words or morphemes (they are called affixes). The study of the rules for the internal structure of words are called **morphology**. For example, stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form, generally a written word form. The rules of the internal structure of phrases and sentences are called **syntax**. Parsing, or syntax analysis, is the process of analysing a string of symbols, either in natural language, computer languages or data structures, conforming to the rules of a formal grammar. It involves breaking down a text into its component parts of speech with an explanation of the form, function, and syntactic relationship of each part. This is mainly determined from the study of conjugations and declensions.

[5] An entity is something that exists apart from other things, having its own independent existence

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.5.1.2 Representational systems for NLP

Druckman et al. [1988] explained that when people are engaged in activities, they are also making use of a representational system, that is, they are using some internal representation of the materials they are involved with, such as a conversation, a rifle shot, a spelling task. These representations can be visual, auditory, kinesthetic, or involve the other senses. In addition, a person may be creating a representation or recalling one. For example, a person asked to spell a word may visualise that word printed on a piece of paper, may hear it being sounded out, or may construct the spelling from the application of a series of logical rules. Hence, mental processing of events and memories can be treated as if performed by the five senses. This observation led to the concept of a preferred representational system (PRS), the classification of people into fixed visual, auditory or kinesthetic stereotypes (see Grinder et al. [1976]). It has been assumed that in most circumstances and for most people, three of the five sensory based modes seem to dominate in mental processing:

- visual thoughts (V): sight, mental imagery, spatial awareness
- auditory (or linguistic) thoughts (A): sound, speech, dialog, white noise
- kinesthetic (or proprioceptive) sense (K): somatic feelings in the body, temperature, pressure, and also emotion.

The other two senses, gustatory (taste) and olfactory (smell), which are closely associated, often seem to be less significant in general mental processing, and are often considered jointly as one. As a result, one often sees the term VAK in NLP reference texts, to signify these three primary representational systems, as well as the term 4-tuple VAKOG when including all senses. The same term is also known as First Access (see Grinder et al. [2001]) or primary experience (Freud). Representational systems are one of the foundational ideas of NLP and form the basis of many NLP techniques and methods. They are relevant since some tasks are better performed within one representational system than by another.

1.5.1.3 A brief history

In the early days, many language-processing systems were designed by hand-coding a set of rules, that is, by writing grammars or devising heuristic rules for stemming. However, this is rarely robust to natural language variation. Since the statistical revolution in the late 1980s and mid 1990s (see Johnson [2009]), much natural language processing research has relied heavily on machine learning. The machine-learning paradigm calls instead for using statistical inference to automatically learn such rules through the analysis of large corpora [6] of typical real-world examples. Increasingly, research has focused on statistical models, which make soft, probabilistic decisions based on attaching real-valued weights to each input feature. Such models have the advantage that they can express the relative certainty of many different possible answers rather than only one, producing more reliable results when such a model is included as a component of a larger system.
Systems based on machine-learning algorithms have many advantages over hand-produced rules such as:

- The learning procedures used during machine learning automatically focus on the most common cases, whereas when writing rules by hand it is often not at all obvious where the effort should be directed.
- Automatic learning procedures can make use of statistical-inference algorithms to produce models that are robust to unfamiliar input (such as containing words or structures that have not been seen before) and to erroneous input (with misspelled words or words accidentally omitted). Generally, handling such input gracefully with hand-written rules, or, more generally, creating systems of hand-written rules that make soft decisions, is extremely difficult, error-prone and time-consuming.

[6] A corpus (plural corpora) is a set of documents, possibly with human or computer annotations.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Systems based on automatically learning the rules can be made more accurate simply by supplying more input data. However, systems based on hand-written rules can only be made more accurate by increasing the complexity of the rules, which is a much more difficult task. In particular, there is a limit to the complexity of systems based on hand-crafted rules, beyond which the systems become more and more unmanageable. However, creating more data to input to machine-learning systems simply requires a corresponding increase in the number of man-hours worked, generally without significant increases in the complexity of the annotation process.

Recent research has increasingly focused on unsupervised and semi-supervised learning algorithms. Such algorithms are able to learn from data that has not been hand-annotated with the desired answers, or using a combination of annotated and non-annotated data. Generally, this task is much more difficult than supervised learning, and typically produces less accurate results for a given amount of input data. However, there is an enormous amount of non-annotated data available (including, among other things, the entire content of the World Wide Web), which can often make up for the inferior results if the algorithm used has a low enough time complexity to be practical.
In the 2010s, representation learning and deep neural network-style machine learning methods became widespread in natural language processing, achieving state-of-the-art results in many natural language tasks, such as language modelling (see Jozefowicz et al. [2016]), parsing (see Vinyals et al. [2015c]), and many others. Popular techniques include the use of word embeddings to capture semantic properties of words (see Collobert et al. [2008]), and an increase in end-to-end learning of a higher-level task (such as question answering, see Rajpurkar et al. [2016]) instead of relying on a pipeline of separate intermediate tasks (such as part-of-speech tagging and dependency parsing). In some areas, this shift has entailed substantial changes in how NLP systems are designed, such that deep neural network-based approaches may be viewed as a new paradigm distinct from statistical natural language processing. For instance, the term Neural Machine Translation (NMT) emphasises the fact that deep learning-based approaches to machine translation directly learn sequence-to-sequence (seq2seq) transformations (see Sutskever et al. [2014]) (see Section (12.6)), obviating the need for intermediate steps such as word alignment and language modelling that were used in statistical machine translation (SMT). Attention mechanisms allow modelling of dependencies, even if they are distant in the input or output sequences. These attention mechanisms are used with recurrent networks (see Bahdanau et al. [2015], Kim et al. [2017]). Vaswani et al. [2017] proposed a simple network architecture, called the Transformer, solely based on attention mechanism, with no recurrence and convolutions. Results on translation tasks showed superior in quality.

1.5.2 NLP tasks

Some of the most commonly researched tasks in natural language processing are listed below. Even though the NLP tasks are closely intertwined, they are frequently subdivided into categories for convenience.

1.5.2.1 Syntax

As discussed in Section (1.5.1), the syntax characterises the rules of the internal structure of phrases and sentences. The syntax category includes:

- Lexing or tokenisation: A lexical token, or simply **token**, is **a string with an assigned and identified meaning**. It is structured as a pair consisting of a token name and an optional token value. The token name is a category of lexical unit. Some common token names are:
  - identifier: names the programmer chooses (x, colour, Up).
  - keyword: names already in the programming language (if, while, return).
  - separator (also known as punctuators): punctuation characters and paired-delimiters ($\{, (, ;$)
  - operator: symbols that operate on arguments and produce results ($+, <, =$).
  - literal: numeric, logical, textual, reference literals (true, "music").

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

  - comment: line, block (**// must be negative**).

  We also distinguish special types:

  - Bigrams: tokens of two consecutive written words.
  - Trigrams: tokens of three consecutive written words.
  - N-grams: tokens of any number N of consecutive written words.

  Tokenisation is the process of demarcating and possibly classifying sections of a string of input characters. The resulting tokens are then passed on to some other form of processing. The process can be considered a subtask of parsing input. Lexers are generally quite simple, with most of the complexity deferred to the parser or semantic analysis phases.

- Stemming: It is the process of reducing inflected (or sometimes derived) words to their word stem, base, or root form. Simple stemmers look up the inflected form in a lookup table, making them fast and efficient. This approach may use preliminary part-of-speech tagging to avoid overstemming. Lemmatisation (see below) is a more complex approach involving first determining the part of speech of a word, and applying different normalisation rules for each part of speech. The part of speech is first detected prior to attempting to find the root since for some languages, the stemming rules change depending on a word's part of speech. Stochastic algorithms involve using probability to identify the root form of a word. They are trained on a table of root form to inflected form relations to develop a probabilistic model.

- Lemmatisation: In many languages, words appear in several inflected forms. The base form, that one might look up in a dictionary, is called the lemma for the word. Lemmatisation is the algorithmic process of determining the lemma of a word based on its intended meaning. Unlike stemming, lemmatisation depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighbouring sentences or even an entire document. In presence of straightforward inflected forms, lemmatisation uses a simple dictionary lookup. In more complex forms a rule-based system is required. Such rules can be either hand-crafted or learned automatically from an annotated corpus.

- Morphological segmentation: Separate words into individual morphemes and identify the class of the morphemes. The difficulty of this task depends greatly on the complexity of the morphology [7] of the language being considered. English has fairly simple morphology, especially inflectional morphology, so that it is often possible to ignore this task entirely and simply model all possible forms of a word (for example "open, opens, opened, opening") as separate words. However, in languages such as Turkish or Meitei, a highly agglutinated Indian language, such an approach is not possible, as each dictionary entry has thousands of possible word forms.

- Part-of-speech (POS) tagging: Given a sentence, determine the part of speech for each word. Many words, especially common ones, can serve as multiple parts of speech. For example, "book" can be a noun ("the book on the table") or verb ("to book a flight"); further "set" can be a noun, verb or adjective; and "out" can be any of at least five different parts of speech. Some languages have more such ambiguity than others. Languages with little inflectional morphology, such as English, are particularly prone to such ambiguity. Chinese is prone to such ambiguity because it is a tonal language during verbalization. Such inflection is not readily conveyed via the entities employed within the orthography to convey intended meaning.

- Grammar induction: Generate a formal grammar that describes a language's syntax. It learns a formal grammar from a set of observations, thus constructing a model which accounts for the characteristics of the observed objects. More generally, grammatical inference is that branch of machine learning where the instance space consists of discrete combinatorial objects such as strings, trees and graphs.

[7] Morphology is the structure of words.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Parsing: It refers to the formal analysis by a computer of a sentence or other string of words into its constituents, resulting in a parse tree showing their syntactic relation to each other, which may also contain semantic and other information. This is illustrated in Figure (1.5).

Figure 1.5: Parse tree.

The grammar for natural languages is ambiguous and typical sentences have multiple possible analyses. In fact, perhaps surprisingly, for a typical sentence there may be thousands of potential parses (most of which will seem completely nonsensical to a human). As a result, some parsing algorithms may generate a parse forest or list of parse trees for a syntactically ambiguous input. There are two primary types of parsing, Dependency Parsing and Constituency Parsing. Dependency Parsing focuses on the relationships between words in a sentence (marking things like Primary Objects and predicates), whereas Constituency Parsing focuses on building out the Parse Tree using a Probabilistic Context-Free Grammar (PCFG).

- Sentence breaking (also known as sentence boundary disambiguation): Given a chunk of text, find the sentence boundaries. Sentence boundaries are often marked by periods or other punctuation marks, but these same characters can serve other purposes (for example marking abbreviations).

- Word segmentation: Separate a chunk of continuous text into separate words. For a language like English, this is fairly trivial, since words are usually separated by spaces. However, some written languages like Chinese, Japanese and Thai do not mark word boundaries in such a fashion, and in those languages text segmentation is a significant task requiring knowledge of the vocabulary and morphology of words in the language. Sometimes this process is also used in cases like Bag of Words (BOW) creation in data mining.

- Terminology extraction: The goal is to automatically extract relevant terms from a given corpus. In general, approaches to automatic term extraction make use of linguistic processors (part of speech tagging, phrase chunking) to extract terminological candidates, that is, syntactically plausible terminological noun phrases.

1.5.2.2 Semantics

As discussed in Section (1.5.1), semantics characterises the meaning that is connected to individual signs, morphemes, words, phrases, and texts. The semantics category includes:

- Word embedding: It is any of a set of language modelling and feature learning techniques in NLP where words or phrases from the vocabulary are mapped to vectors of real numbers (see Fisher [1957]). Conceptually, it

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

involves a mathematical embedding from a space with many dimensions per word to a continuous vector space with a much lower dimension. This is illustrated in Figure (1.6). It aims at quantifying and categorising semantic similarities between linguistic items based on their distributional properties in large samples of language data. There exists a semantic space with lexical items (words or multi-word terms) represented as vectors or embeddings. Since vector space models for words and their distributional data results in a very sparse vector space of high dimensionality, one must therefore perform some dimension reduction of words representations in contexts by learning a distributed representation for words.

Figure 1.6: Word vectorisation.

- Lexical semantics (or lexicosemantics): Lexical items contain information about category (lexical and syntactic), form and meaning. Lexical semantics looks at how the meaning of the lexical units correlates with the structure of the language or syntax. What is the computational meaning of individual words in context? This is referred to as syntax-semantic interface. The study of lexical semantics looks at: (1) the classification and decomposition of lexical items, (2) the differences and similarities in lexical semantic structure cross-linguistically, (3) the relationship of lexical meaning to sentence meaning and syntax.

- Machine translation: It automatically translates text from one human language to another. This is one of the most difficult problems, and is a member of a class of problems colloquially termed "AI-complete", that is, requiring all of the different types of knowledge that humans possess (grammar, semantics, facts about the real world, and more) in order to solve properly.

- Named entity recognition (NER): Given a stream of text, determine which items in the text map to proper names, such as people or places, and what the type of each such name is (for example person, location, organisation). Note that, although capitalisation can aid in recognising named entities in languages such as English, this information cannot aid in determining the type of named entity, and in any case is often inaccurate or insufficient. For example, the first letter of a sentence is also capitalised, and named entities often span several words, only some of which are capitalised. Furthermore, many other languages in non-Western scripts (such as Chinese or Arabic) do not have any capitalisation at all, and even languages with capitalisation may not consistently use it to distinguish names. For example, German capitalises all nouns, regardless of whether they are names, and French and Spanish do not capitalise names that serve as adjectives.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Natural language understanding (NLU): It converts chunks of text into more formal representations such as first-order logic structures that are easier for computer programs to manipulate. NLU involves the identification of the intended semantic from the multiple possible semantics which can be derived from a natural language expression which usually takes the form of organised notations of natural language concepts. Introduction and creation of language metamodel and ontology are efficient however empirical solutions. An explicit formalisation of natural language semantics without confusions with implicit assumptions such as closed-world assumption (CWA) vs. open-world assumption, or subjective Yes/No vs. objective True/False is expected for the construction of a basis of semantics formalisation.

- Optical character recognition (OCR): Given an image representing printed text, determine the corresponding text. See Section (1.4.1 and (1.4.4.2)).

- Question answering: Given a human-language question, determine its answer (see Rajpurkar et al. [2016]). Typical questions have a specific right answer (such as "What is the capital of Spain?"), but sometimes open-ended questions are also considered (such as "What is the meaning of life?"). Recent works have looked at even more complex questions.

- Natural language generation (NLG): It converts information from computer databases or semantic intents into readable human language. It aims to generate natural language sentences conditioned on some inputs, including tasks like neural machine translation (NMT), text summarisation, and conversational response generation.

- Recognising textual entailment: It is a directional relation between text fragments where he entailing and entailed texts are termed **text (t)** and **hypothesis (h)**, respectively. The relation holds whenever the truth of one text fragment follows from another text. That is, given two text fragments, determine if one being true entails the other, entails the other's negation, or allows the other to be either true or false.

- Relationship extraction: It is the detection and classification of semantic relationship mentions within a set of artifacts, typically from text or XML documents. That is, given a chunk of text, identify the relationships among named entities (for example who is married to whom). One solution to this problem make use of domain ontologies. Another approach involves visual detection of meaningful relationships in parametric values of objects listed on a data table that shift positions as the table is permuted automatically as controlled by the software user. However, the poor coverage, rarity and development cost related to structured resources such as semantic lexicons (such as WordNet, UMLS) and domain ontologies (such as the Gene Ontology) has given rise to new approaches based on broad, dynamic background knowledge on the Web. For instance, the ARCHILES technique (see Wong et al. [2009]) uses only Wikipedia and search engine page count for acquiring coarse-grained relations to construct lightweight ontologies. Recently, Nguyen et al. [2019] proposed end-to-end systems which jointly learn to extract entity mentions and their semantic relations with strong potential to obtain high performance.

- Semantic search: It denotes search with meaning, as opposed from lexical search where the search engine looks for literal matches of the query words or variants of them, without understanding the overall meaning of the query (see Bast et al. [2016]). It seeks to improve search accuracy by understanding the searcher's intent and the contextual meaning of terms as they appear in the searchable data-space, whether on the Web or within a closed system, to generate more relevant results.

- Sentiment analysis (see also multimodal sentiment analysis): It extracts subjective information usually from a set of documents, often using online reviews to determine "polarity" about specific objects. It is especially useful for identifying trends of public opinion in the social media, for the purpose of marketing (see Socher et al. [2013]).

- Topic segmentation and recognition: Given a chunk of text, separate it into segments each of which is devoted to a topic, and identify the topic of the segment.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Word sense disambiguation: Many words have more than one meaning and we want to select the meaning which makes the most sense in context. For this problem, we are typically given a list of words and associated word senses from a dictionary or from an online resource such as WordNet.

1.5.2.3 Discourse

Discourse identifies and describes written and spoken communications. In semantics, it is a conceptual generalisation of conversation within each modality and context of communication. Thus, the term is studied in corpus linguistics. The discourse category includes:

- Automatic summarisation: Produce a readable summary of a chunk of text. Often used to provide summaries of text of a known type, such as articles in the financial section of a newspaper.
- Coreference resolution: Given a sentence or larger chunk of text, determine which words ("mentions") refer to the same objects ("entities"). Anaphora resolution is a specific example of this task, and is specifically concerned with matching up pronouns with the nouns or names to which they refer. The more general task of coreference resolution also includes identifying "bridging relationships" involving referring expressions. For example, in a sentence such as "He entered Paul's house through the back door", "the back door" is a referring expression and the bridging relationship to be identified is the fact that the door being referred to is the back door of Paul's house (rather than of some other structure that might also be referred to).
- Discourse analysis: It includes a number of related tasks. One of them being to identify the discourse structure of connected text, that is, the nature of the discourse relationships between sentences (such as elaboration, explanation, contrast). Another one is to recognise and classify the speech acts in a chunk of text (such as yes-no question, content question, statement, assertion).

1.5.2.4 Speech

Speech is human vocal communication using language. Speech production involves the selection of appropriate words and the appropriate form of those words from the lexicon and morphology, and the organisation of those words through the syntax. Then, the phonetic properties of the words are retrieved and the sentence is articulated through the articulations associated with those phonetic properties (see Levelt [1999]). The speech category includes:

- Speech recognition: Given a sound clip of a person or people speaking, it determines the textual representation of the speech. This is the opposite of text to speech and is one of the extremely difficult problems colloquially termed AI-complete. In natural speech there are hardly any pauses between successive words, and thus speech segmentation is a necessary subtask of speech recognition. Note also that in most spoken languages, the sounds representing successive letters blend into each other in a process termed coarticulation, so the conversion of the analog signal to discrete characters can be a very difficult process. Further, given that words in the same language are spoken by people with different accents, the speech recognition software must be able to recognise the wide variety of input as being identical to each other in terms of its textual equivalent.
- Speech segmentation: Given a sound clip of a person or people speaking, separate it into words. A subtask of speech recognition and typically grouped with it.
- Text-to-speech (TTS): Given a language text, transform those units and produce a spoken representation. Text-to-speech can be used to aid the visually impaired. A TTS system is composed of two parts:
    1. A front-end: First, it converts raw text containing symbols like numbers and abbreviations into the equivalent of written-out words. This process is often called text normalisation, pre-processing, or tokenisation. Then, it assigns phonetic transcriptions to each word, and divides and marks the text into prosodic units, like phrases, clauses, and sentences. This process is called text-to-phoneme or grapheme-to-phoneme

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

conversion. Phonetic transcriptions and prosody information together make up the symbolic linguistic representation that is output by the front-end.

2. A back-end (or synthesiser): It converts the symbolic linguistic representation into sound. In certain systems, this part includes the computation of the target prosody (pitch contour, phoneme durations), which is then imposed on the output speech.

This is illustrated in Figure (1.7).

Figure 1.7: TTS system.

1.5.3 Some example tasks

In this section we discuss a few NLP tasks.

1.5.3.1 OCR

Optical character recognition (OCR), is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example from a television broadcast). It is a common method of digitising printed texts so that they can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes such as cognitive computing, machine translation, (extracted) text-to-speech, key data and text mining (see Schantz [1982]). OCR is a field of research in pattern recognition, artificial intelligence and computer vision.

The types are as follows:

- Optical character recognition (OCR): targets typewritten text, one glyph [8] or character at a time.
- Optical word recognition: targets typewritten text, one word at a time (for languages that use a space as a word divider).
- Intelligent character recognition (ICR): also targets handwritten printscript or cursive text one glyph or character at a time, usually involving machine learning.
- Intelligent word recognition (IWR): also targets handwritten printscript or cursive text, one word at a time. This is especially useful for languages where glyphs are not separated in cursive script.

[8] In typography, a glyph is an elemental symbol within an agreed set of symbols, intended to represent a readable character for the purposes of writing. Glyphs are considered to be unique marks that collectively add up to the spelling of a word or contribute to a specific meaning of what is written, with that meaning dependent on cultural and social usage.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Optical character recognition software takes several steps to convert an image file into an editable document. Each step in this process uses a specific algorithm to alter, enhance, and interpret the images found within a file. Each and every step involved in this process is critical to the overall success of OCR. Smallest errors will cause major issues, resulting in a poorly translated final document.

1. Loading the image file: In order for OCR to be effective, it must support a wide array of file formats, including PDF, BMP, TIFF, JPEG, and PNG files. These files can be scanned documents, photographs, or even read-only files. Regardless of the original format, OCR software will transform these files into easily accessible and editable data.

2. Pre-processing:

   - Improving image quality and orientation: Depending on the method in which the image file was created, there are a number of issues that may arise. In general, an image file might be skewed or contain noise (a/k/a varying brightness or colour). The software will work to de-skew, remove any noise, and improve the overall quality of the images. This is a critical step, since blurry or skewed images are not interpreted properly.
   - Removing lines: Lines can prove to be disastrous when interpreting characters. In order to remain as accurate as possible, lines are detected and removed. This allows for better recognition quality when converting tables, underlined words, and more. The removal of lines will ensure that characters are recognised accurately.
   - Analysing the page: During this stage of OCR, the layout of the original file is noted and processed. This includes the detection of text positions, white space, and the prioritisation of important text areas or sections.
   - Detecting words and lines of text: This is the beginning stage of actual character recognition. The software begins to identify individual words and entire lines of data. This is a critical pre-process for properly recognising characters as it sets the stage for the analysis and correction of broken or merged characters.
   - Analysing and fixing of broken or merged characters: Depending on the quality of the original file, there are often errors in which characters are broken or blurred together. The OCR software must now break down and resolve these errors in order to properly interpret the appropriate characters.
   - Normalisation: Normalise aspect ratio and scale.

3. Recognising characters: This is the primary function of OCR. Now that the original file has been processed, cleaned, and fixed the OCR technology can begin to read and translate characters. Each image of every character is converted into a character code. If the algorithm is unsure of a character the software will produce multiple character codes and choose the proper character later on. There are two basic types of core OCR algorithm, which may produce a ranked list of candidate characters:

   (a) Matrix matching (or pattern) involves comparing an image to a stored glyph on a pixel-by-pixel basis.
   (b) Feature extraction decomposes glyphs into features like lines, closed loops, line direction, and line intersections. The extraction features reduces the dimensionality of the representation and makes the recognition process computationally efficient.

4. Post-processing: OCR accuracy can be increased if the output is constrained by a lexicon, a list of words that are allowed to occur in a document. For example, it can be all the words in the English language, or a more technical lexicon for a specific field. However, this technique can be problematic if the document contains words not in the lexicon, like proper nouns. Tesseract uses its dictionary to influence the character segmentation step, for improved accuracy.

5. Saving the file: After the file has been fully interpreted, it can be saved to the desired file format.

These steps make up the primary processes involved in Optical Character Recognition.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.5.3.2 Named-entity recognition

In information extraction, a named entity is a real-world object, such as persons, locations, organisations, products, etc., that can be denoted with a proper name. It can be abstract or have a physical existence. In the expression "Named Entity", the word "Named" aims to restrict the possible set of entities to only those for which one or many rigid designators stands for the referent. A designator is rigid when it designates the same thing in every possible world. On the other hand, flaccid designators may designate different things in different possible worlds. Named-entity recognition (NER) (also known as entity identification, entity chunking and entity extraction) is a subtask of information extraction that seeks to locate and classify named entity mentions in unstructured text into pre-defined categories such as the person names, organisations, locations, medical codes, time expressions, quantities, monetary values, percentages.
In the expression named entity, the word named restricts the task to those entities for which one or many strings, such as words or phrases, stands (fairly) consistently for some referent. While closely related to rigid designators (see Kripke [1971]), NER deals with many names and referents that are not philosophically rigid. Rigid designators include proper names as well as terms for certain biological species and substances, but exclude pronouns (such as "it"), descriptions that pick out a referent by its properties (see also De dicto and de re), and names for kinds of things as opposed to individuals (for example "Bank").
Full named-entity recognition is often broken down, conceptually as well as in implementations, in two distinct problems: detection of names, and classification of the names by the type of entity they refer to (such as person, organisation, location and other). The first phase is typically simplified to a segmentation problem: names are defined to be contiguous spans of tokens, with no nesting, so that "Bank of Canada" is a single name, disregarding the fact that inside this name, the substring "Canada" is itself a name. This segmentation problem is formally similar to chunking. The second phase requires choosing an ontology by which to organise categories of things.
Temporal expressions and some numerical expressions (such as money, percentages) may also be considered as named entities in the context of the NER task. While some instances of these types are good examples of rigid designators (for example the year 1970) there are also many invalid ones (for example, I take my vacations in "July"). In the first case, the year 1970 refers to the 1970th year of the Gregorian calendar, while in the second case, the month July may refer to the month of an undefined year (past July, next July, every July). It is arguable that the named entity definition is loosened in such cases for practical reasons. Hence, the definition of the term named entity is not strict and often has to be explained in the context in which it is used. Certain hierarchies of named entity types have been proposed in the literature. For instance, Ritter et al. [2011] used a hierarchy based on common Freebase entity types in ground-breaking experiments on NER over social media text.
Several measures have been defined for evaluating the quality of a NER system's output. We can consider the accuracy on the token level, but it suffers from two problems: (1) the vast majority of tokens in real-world text are not part of entity names as usually defined, so the baseline accuracy (always predict "not an entity") is extravagantly high, typically $> 90\%$; and (2) mispredicting the full span of an entity name is not properly penalised (finding only a person's first name when their last name follows is scored as $\frac{1}{2}$ accuracy).
Tjong Kim Sang et al. [2003] proposed a variant of the $F1$ which is defined as follows:

- Precision is the number of predicted entity name spans that line up exactly with spans in the gold standard evaluation data. That is, when [Person Hans] [Person Blick] is predicted but [Person Hans Blick] was required, precision for the predicted name is zero. Precision is then averaged over all predicted entity names.
- Recall is similarly the number of names in the gold standard that appear at exactly the same location in the predictions.
- $F1$ score is the harmonic mean of these two.

In that setting, any prediction that misses a single token, includes a spurious token, or has the wrong class, is a hard error and does not contribute to either precision or recall.
Alternatiely, Esuli et al. [2010] proposed evaluation models based on a token-by-token matching. Such models are able to handle also partially overlapping matches, yet fully rewarding only exact matches. They allow a finer

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

grained evaluation and comparison of extraction systems, taking into account also the degree of mismatch in non-exact predictions.
Note, NER systems have been created that use linguistic grammar-based techniques as well as statistical models such as machine learning. Even though hand-crafted grammar-based systems typically obtain better precision, this is obtained at the cost of lower recall and months of work by experienced computational linguists. Similarly, statistical NER systems typically require a large amount of manually annotated training data. Semisupervised approaches have been suggested to avoid part of the annotation effort (see Nothman et al. [2013]). Some authors considered using different classifier types to perform machine-learned NER, with conditional random fields being a typical choice (see Finkel et al. [2005]).
In general, NER systems developed for one domain do not typically perform well on other domains, and considerable effort is involved in tuning NER systems to perform well in a new domain. This is true for both rule-based and trainable statistical systems. Despite the high $F1$ numbers reported on the MUC-7 dataset, the problem of NER is far from being solved. The main efforts are directed to reducing the annotation labour by employing semi-supervised learning (see Turian et al. [2010]), robust performance across domains, and scaling up to fine-grained entity types. Comparisons were made about the NER performances from different statistical models such as HMM (Hidden Markov Model), ME (Maximum Entropy), and CRF (Conditional Random Fields) and feature sets. Some researchers recently proposed graph-based semi-supervised learning model for language specific NER tasks (see Han et al. [2015]).

1.5.3.3 Paraphrasing and textual entailment

A characteristic of natural language is that there are many different ways to state what one wants to say: several meanings can be contained in a single text and that the same meaning can be expressed by different texts. This variability of semantic expression can be seen as the dual problem of language ambiguity. Together, they result in a many-to-many mapping between language expressions and meanings. This is leading to ambiguity including

- Lexical ambiguity: For example "She is looking for a *match*" and "The fisherman went to the *bank*".
- Syntactic ambiguity: For example "The chicken is ready to eat" and "I saw the man with binoculars".
- Referential ambiguity: For example "The boy told his father the theft, He was very upset".

The task of paraphrasing involves recognising when two texts have the same meaning and creating a similar or shorter text that conveys almost the same information. Textual entailment (TE) measures natural language understanding as it asks for a semantic interpretation of the text. It is similar to paraphrasing, but weakens the relationship to be unidirectional (see Androutsopoulos et al. [2010]). Mathematical solutions to establish textual entailment can be based on the directional property of this relation, by making a comparison between some directional similarities of the texts involved. Many approaches and refinements of approaches have been considered, such as word embedding, logical models, graphical models, rule systems, contextual focusing, and machine learning. Note, practical or large-scale solutions avoid these complex methods and instead use only surface syntax or lexical relationships, but are correspondingly less accurate.

1.5.3.4 Natural language understanding

Natural-language understanding (NLU) is a subtopic of natural-language processing in artificial intelligence that deals with machine reading comprehension. It is considered an AI-hard problem. It is the post-processing of text, after the use of NLP algorithms, that utilises context from recognition devices (Automatic Speech Recognition [ASR], vision recognition, last conversation, misrecognised words from ASR, personalised profiles, microphone proximity), in all of its forms, to discern meaning of fragmented and run-on sentences to execute an intent from typically voice commands. NLU consists of the following:

- It has an ontology around particular product vertical that is used to figure out the probability of an intent.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- It also has a defined list of known intents that derives the message payload from designated contextual information recognition sources.
- It will provide back multiple message outputs to separate services (software) or resources (hardware) from a single derived intent (response to voice command initiator with visual sentence (shown or spoken) and transformed voice command message too different output messages to be consumed for M2M communications and actions).

Most natural-language-understanding systems share some common components:

- The system needs a lexicon of the language and a parser and grammar rules to break sentences into an internal representation. The construction of a rich lexicon with a suitable ontology requires significant effort. For instance, the Wordnet lexicon required many person-years of effort (see Miller et al. [1990]).

- The system also needs a semantic theory to guide the comprehension. The interpretation capabilities of a language-understanding system depend on the semantic theory it uses. Competing semantic theories of language have specific trade-offs in their suitability as the basis of computer-automated semantic interpretation (see Lawler et al [1998]). These range from naive semantics or stochastic semantic analysis to the use of pragmatics to derive meaning from context (see Green [1996]). Semantic parsers convert natural-language texts into formal meaning representations (Wong et al. [2006]).

Advanced applications of NLU also attempt at incorporating logical inference within their framework, which is generally achieved by mapping the derived meaning into a set of assertions in predicate logic, then using logical deduction to arrive at conclusions. Thus, systems based on functional languages such as Lisp need to include a subsystem to represent logical assertions, while logic-oriented systems such as those using the language Prolog generally rely on an extension of the built-in logical representation framework.

1.5.4 Contextual word representations

Smith [2020] wrote a contextual introduction on how one puts words into computers, focussing on word vectors (also known as word embeddings) and contextual word vectors.

1.5.4.1 Word vectors

The simplest representation of a piece of text is a sequence of characters (a single byte or several). One can roughly represents words in two ways:

1. Word token: It is a word observed in a piece of text. In some languages, identifying the boundaries of the word tokens is a complicated procedure (and speakers of the language may not agree on the "correct" rules for splitting text into words). However, in English, we tend to use whitespace and punctuation to delimit words, which is known as tokenization, is "solved". For example, the first sentence of this paragraph is typically tokenized as follows (with the end-of-sentence punctuation treated as a separate token):

   A word token is a word observed in a piece of text.

   There are 13 tokens in the sentence.

2. Word type: It is a distinct word, in the abstract, rather than a specific instance. Every word token is said to "belong" to its type. In the above example, there are only 11 word types, since the two instances of word share the same type, as do the two instances of a. When we count the occurrences of a vocabulary word in a collection of texts (known as a corpus, plural corpora), we are counting the tokens that belong to the same word type. A word type can be represented as a string.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Words can be integerised [9], so that each word type is given a unique (and more or less arbitrary) non-negative integer value. However, it is difficult to manipulate discrete word types when words have similar properties. When it comes to similarity: anything it discovers about one word should transfer to similar words. This is the definition of mathematical representation in Section (1.1.1.4). There are two approaches regarding bringing this information about similarity into computer programs:

1. The information is known by humans and can be explicitly encoded into programs and accessed as needed. For examples, WordNet is a lexical database that stores words and relationships among them such as synonymy and hyponymy. It also explicitly captures the different senses of words that take multiple meanings.

2. The information resides in artifacts such as text corpora, and we can use a separate set of programs to collect and suitably organize the information for use in NLP.

Combining these two approaches, we can derive a notion of a word type as a vector instead of an integer (Representation theory). Hence, we can choose the dimensionality of the vector and allocate different dimensions for different purposes. There are numerous ways of allocating **dimensions** to vectors representing word types (either syntactic, like "verb", or semantic, like "animate"), or to multiword sequences (for example, "White House and hot dog"). The technical term used for these dimensions is **features**. Features can be designed by experts, or they can be derived using automated algorithms.

It has been argued that words (or expressions) that can be used in similar ways are likely to have related meanings. For instance, considering the "week example", we can collect information about the ways a word type w is used, by counting the number of times it appears near every other word type. Looking at the full **distribution of contexts** (nearby words or sequences of words) in a corpus where w is found, corresponds to taking a distributional view of word meaning. One highly successful approach to automatically deriving features based on this idea is clustering (see Section (3.3.4)). For instance, Brown et al. [1992] proposed a clustering algorithm that automatically organise words into clusters based on the contexts they appear in, in a corpus. Words occurring in the same neighbouring contexts (other words) are grouped together into a cluster. These clusters can then be merged into larger clusters, resulting in interpretable hierarchy. It can be rebuilt using any given corpus, so that every word observed can be included. Hence, suitable word clusters could be built separately for news text, or biomedical articles, or tweets. Another approach started by creating word vectors in which each **dimension** corresponded to the **frequency** the word type occurred in some context (see Deerwester et al. [1990]). As an example, one dimension might correspond to the and contain the number of times the word occurred within a small window of a token the. Contextual patterns on the left or the right, and of varying distances and lengths, might be included. The result is a vector perhaps many times longer than the size of the vocabulary, in which each dimension contains a tiny bit of **information** that may or may not be useful. Dimensionality reduction techniques can be used to compress these vectors into shorter vectors in which redundancies across dimensions are collapsed. The dimensionality can be chosen by the NLP programmer to suit the needs of the program. More compact vectors might be more efficient to compute with, and might also benefit from the lossiness of the compression, since corpus-specific noise might fall away. However, there is a trade-off: longer, less heavily compressed vectors retain more of the original information in the distributional vectors. While the individual dimensions of the compressed vectors are not easily interpreted, we can use well-known algorithms to find a word's nearest neighbours in the vector space, and these are often found to be semantically related words, as one might hope. This led to the development of vector space semantics (see Turney et al. [2010]), in which arithmetic operations were applied to word vectors to probe what kind of "meanings" had been learned. Famously, analogies like "man is to woman as king is to queen" led to testing whether $v(man) - v(woman) = v(king) - v(queen)$. However, one problem with reduced-dimensionality vectors is that the individual dimensions are no longer interpretable features that can be mapped back to intuitive building blocks contributing to the word's meaning. The latter is distributed across the whole vector. Hence, these vectors are sometimes called distributed representations.

[9] Represented as integer

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

As corpora grew, scalability became a challenge, since the number of observable contexts grew as well. Underlying all word vector algorithms is the notion that the value placed in each **dimension** of each word type's vector is a **parameter** that will be optimised, alongside all the other parameters, to best fit the observed patterns of the words in the data. Since these parameters are continuous values, the notion of "fitting the data" can be operationalised as a smooth, continuous objective function. As a result, selecting the parameter values is done using iterative algorithms based on gradient descent (see Section (5.1.4)). Using tools that had become popular in machine learning, faster methods based on stochastic optimisation developed, such as the word2vec package (see Mikolov et al. [2013]). This led researchers with large corpora and powerful computing infrastructure to construct word vectors using an established (often expensive) iterative method, and then publish the vectors for anyone to use. Numerous ideas developed to find methods for obtaining distributional word vectors such as

- First map each input word token to its vector, and then "feed" the word vectors into the neural network model, which performs a task like translation. The vectors can be fixed in advance (or pre-trained from a corpus, using methods like those above, often executed by someone else), or they can be treated as parameters of the neural network model, and adapted to the task specifically (see Collobert et al. [2011]). Fine-tuning refers to initialising the word vectors by pre-training, then adapting them through task-specific learning algorithms (see Section (1.5.5)). The word vectors can also be initialised to random values, then estimated solely through task learning, which we might call "learning from scratch".
- Use expert-built data structures like WordNet as additional input to creating word vectors. One approach, retrofitting, starts with word vectors extracted from a corpus, then seeks to automatically adjust them so that word types that are related in WordNet are closer to each other in vector space (see Faruqui et al. [2016]).
- Use bilingual dictionaries to align the vectors for words in two languages into a single vector space, so that, for example, the vectors for the English word type cucumber and the French word type concombre have a small Euclidean distance (see Faruqui et al. [2014]). By constructing a function that reorients all the English vectors into the French space (or vice versa), researchers hoped to align all the English and French words, not just the ones in the bilingual dictionary.
- Calculate a word's vectors in part (or in whole) from its character sequence (see Ling et al. [2015]). These methods make use of neural networks to map arbitrary-length sequences into a fixed-length vector. This is called sequence to sequence (see Section (14.1)). This has two interesting effects: (1) in languages with intricate word formation systems (morphology), variants of the same underlying root may have similar vectors, (2) differently spelled variants of the same word will have similar vectors. This kind of approach was quite successful for social media texts, where there is rich spelling variation.

1.5.4.2 Contextual word vectors

So far, we have assumed that each word type was represented in a computer program by using a fixed data object: first an integer, then a vector. However, we saw that words have different meanings in different contexts. It was captured by experts in crafting WordNet, in which, for example, get is mapped to over 30 different meanings (or senses). It is difficult to obtain widespread agreement on how many senses should be allocated to different words, or on the boundaries between one sense and another since word senses may be fluid. In neural networks, each word token's type vector is passed into a function that transforms it based on the words in its nearby context, giving a new version of the word vector, now specific to the token in its particular context. In the above example defining word token, two instances of a might have different vectors, because one occurs between is and $word$ and the other occurs between in and $piece$. This is illustrated in Figure (1.8) where the left figure is a visualisation of the relationships between the three-word vectors, and the right figure is a hypothetical visualisation of contextual vectors for tokens of astronomers, bodies, and objects.

Hence, by representing word types independent of context, we are solving a problem that is more difficult than it should be. Since words mean different things in different contexts, we are requiring that type representations capture all of the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 1.8: Word to vector representation.

possibilities. Moving to **word token vectors** simplifies things, asking the word token representation to capture only what a word means in this context. For the same reasons the collection of contexts a word type is found in provide clues about its meaning(s), a particular token's context provides clues about its specific meaning. We expect words that are similar to each other to be good substitutes for each other. For instance, ELMo (embeddings from language models) (see Peters et al. [2018]) considered **word token vectors**, that is, vectors for words in context, or contextual word vectors that are pre-trained on large corpora. It works as follows:

- If every word token is going to have its own vector, then the vector should depend on an arbitrarily long context of nearby words. To obtain a **context vector**, we start with **word type vectors**, and pass them through a neural network that can transform arbitrary-length sequences of left- and/or right-context word vectors into a single fixed-length vector. This is called sequence to sequence (see Section (14.1)). Unlike word type vectors, which are essentially lookup tables, **contextual word vectors** are built from both: (1) type-level vectors, and (2) neural network parameters that contextualise each word. ELMo trains one neural network for left contexts (going back to the beginning of the sentence a token appears in) and another neural network for right contexts (up to the end of the sentence). Longer contexts, beyond sentence boundaries, are in principle possible as well.

- We saw above that estimating word vectors required "fitting the data" (here, a corpus) by solving an optimisation problem. A long standing data-fitting problem in NLP is **language modelling**, which refers to predicting the next word given a sequence of "history" words. Further, existing word (type) vector algorithms are based on a notion of fixed-size contexts, collected across all instances of the word type in a corpus. ELMo goes one step further, by using arbitrary-length histories and directly incorporating the language models known at the time to be most effective (based on recurrent neural networks such as LSTM (see Chapter (6)). Here, RNNs are trained as language models, then the output context vectors are used for each word token as pre-trained word (token) vectors.

The powerful idea that text data can shed light on a word's meaning, by observing the contexts in which a word appears, has led researchers to try to capture a word token's meaning primarily through the specific context it appears in. For example, every instance of $plant$ will have a different word vector: those with a context that look like a context for references to vegetation are expected to be close to each other, while those that are likely contexts for references to manufacturing centers will cluster elsewhere in vector space. New ideas in NLP are tested on benchmark tasks with objectively measurable performance scores. ELMo was shown to perform very well in a large number of tests. A similar approach, called ULMFiT, was introduced by Howard et al. [2018], showing a benefit for text classification methods. Introducing several innovations to the learning method, and learned from more data, BERT (bidirectional encoder representations from transformers) proposed by Devlin et al. [2018], achieved a further 45% error reduction (relative to ELMo) on the first task and 7% on the second. BERT pre-training focuses on two tasks: (1) prediction of words given contexts on both sides (rather than one or the other), and (2) predicting the words in a sentence given

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

its preceding sentence. While ELMo is derived from language modelling (see Bengio et al. [2003]), the modelling problem solved by BERT (that is, the objective function minimised during estimation) is rather different. Numerous articles extended this approach, resulting in widely adopted models like GPT-2 (see Radford et al. [2019]), RoBERTa (see Liu et al. [2019d]), XLM (see Lample et al. [2019]), XLNet (see Yang et al. [2019]), among others. The space of possible learners for contextual word vectors has not yet been fully explored. Some of the findings on BERT suggest that the role of fine-tuning may be critical.

1.5.5 Transfer learning

1.5.5.1 Definition

As discussed in Section (1.5.4), **language modelling** requires to learn about syntax, semantics, as well as certain facts about the world. Given enough data, a large number of parameters, and enough computing power, a model can do a reasonable job. As a result, natural language processing is increasingly applied to new tasks, new domains, and new languages. It was suggested in a recent predictive-rate distortion (PRD) analysis of human language (see Hahn et al. [2019]) that human language, and language modelling, had infinite statistical complexity, but that it could be approximated well at lower levels. This observation has two implications:

1. We can obtain good results with comparatively small models.
2. There is a lot of potential for scaling up the models.

One solution is to consider transfer learning which is a means of extracting knowledge from a source setting and applying it to a different target setting. In NLP, it is the process of training a model on a large-scale dataset and then using that pre-trained model to conduct learning for another downstream task (that is, target task) (see Ruder [2019]). We store the knowledge gained in solving the source task in the source domain and apply it to the target task and target domain. This is displayed in Figure (1.9).

Figure 1.9: Transfer learning for NLP.

There are different types of transfer learning common in current NLP which can be roughly classified along three dimensions based on

1. whether the source and target settings deal with the same task

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. the nature of the source and target domains
3. the order in which the tasks are learned

Ruder [2019] adapted the taxonomy of Pan et al. [2010] to the transfer learning on a list of scenarios most commonly encountered in NLP. This taxonomy is displayed in Figure (1.10).

Figure 1.10: Taxonomy for transfer learning for NLP.

1.5.5.2 Pretraining

As discussed in Section (1.5.4), pre-trained language models have been used to achieve state-of-the-art results on a wide range of NLP tasks such as sequence labelling and sentence classification. Pre-training allows a model to capture and learn a variety of linguistic phenomena, such as long-term dependencies and negation, from a large-scale corpus. This knowledge is then used (transferred) to initialise and train another model to perform well on a specific NLP task, such as sentiment classification (see Figure (1.9)). For instance, negation is an important attribute for detecting sentiment polarity from textual information. In addition, negation could also be useful for, say, sarcasm detection, which is one of the most complex, unsolved tasks in NLP. Thus, negation is useful for many NLP tasks. More generally, the pre-trained model possesses some kind of universal properties, which is useful in cases where there is a lack of annotated datasets or language resources which is prevalent in NLP research. In general, one seek to transfer as much knowledge as possible from the source setting to the target task or domain. While this knowledge can take on various forms, depending on the task and data, in most cases it relates to the representations learned by neural network models. In practice, transfer learning has often been shown to achieve similar performance compared to a non-pretrained model with 10x fewer examples or more as can be seen for ULMFiT (see Howard et al. [2018]). Pre-trained representations can generally be improved by jointly increasing the number of model parameters and the amount of pretraining data.

We saw above that, over time, pre-trained word representations have incorporated more context. Early approaches such as word2vec (see Mikolov et al. [2013b], Pennington et al. [2014]) learned a single representation for every word

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

independently of its context. These word vectors are static. Later approaches scaled these representations to sentences and documents (see Le et al. [2014], Conneau et al. [2018]). Current approaches learn word representations that change based on the word's context via **context vectors** (see McCann et al. [2017], Peters et al. [2018]). So far, the knowledge obtained from pre-trained language models, such as in the form of word embeddings, perform significantly well for many NLP tasks. However, this knowledge, in the form of latent features, is not broad or general enough to perform well on the target, or downstream tasks, at hand. Over the last years, state-of-the-art models in NLP have become progressively deeper to remedy this limitation. These language modelling systems conduct deep pre-training of entire models with hierarchical representations or graph-based representations. Some of these pre-trained models include ULMFit (see Howard et al. [2018]), ELMo (see Peters et al. [2018]), GLoMo, and OpenAI transformer (see Radford et al. [2018]). This concept is a shift from the use of unary word embeddings, which have been effectively used to address many NLP tasks over the past few years, to the use of more sophisticated and abstract representations. For example, the ELMo can be described as "pretraining the entire model with deep contextualised representations through stacked neural layers" as opposed to just employing the word embeddings (that is, unary representations) as initialisation. Until recently, the $2-3$ layer deep BiLSTM (see Graves et al. [2005b]) was state-of-the-art, with machine translation being an outlier with 16 layers (see Wu et al. [2016]). In contrast, current models like BERT-Large (see Devlin et al. [2018]) and GPT-2 consist of 24 Transformer blocks (see Vaswani et al. [2017]) and recent models are even deeper.

Contextual word representations (CWRs) have been shown to be predictive of certain linguistic phenomena such as alignments in translation or syntactic hierarchies. This suggests that they are successful at encoding useful and transferable features of language. In general, the network architectures determine what is in a representation. Liu et al. [2019b] studied the representations produced by several recent pre-trained contextualisers (variants of ELMo, the OpenAI transformer language model, and BERT) with a suite of sixteen diverse probing tasks. CWRs are the outputs of a neural network (called a contextualiser) trained on tasks with large datasets. They used probing models (see Belinkov et al. [2019]) to analyse the linguistic information within CWRs. They generated features for words from pre-trained contextualisers and trained a model to make predictions from those features alone. This is illustrated in Figure (1.11).

Figure 1.11: Probing model setup.

The authors investigated the transferability of contextual word representations by quantifying the differences in the transferability of individual layers within contextualisers, especially between recurrent neural networks (RNNs) and transformers. They found that different architectures showed different layer-wise trends in terms of what information they capture.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.5.5.3 Adaptation

There are several approaches for adapting a pre-trained model (source setting) to a target task (domain), such as architectural modifications, optimisation schemes, and whether to obtain more signal.

1.5.5.3.1 Architectural modifications There are two main approaches for architectural modifications:

1. Keep the pre-trained model internals unchanged: One can simply add one or more linear layers on top of a pre-trained model (see BERT by Devlin et al. [2018]). One can also use the model output as input to a separate model, which is often beneficial when a target task requires interactions that are not available in the pre-trained embedding. Examples include span representations or modelling cross-sentence relations.

2. Modify the pre-trained model internal architecture: This is done when one want to adapt to a structurally different target task such as one with several input sequences. In this case, one can use the pre-trained model to initialise as much as possible of a structurally different target task model. We might also want to apply task-specific modifications such as adding skip or residual connections or attention. Alternatively, modifying the target task parameters may reduce the number of parameters that need to be fine-tuned by adding bottleneck modules (adapter modules) between the layers of the pre-trained model (see Houlsby et al. [2019]).

1.5.5.3.2 Optimisation schemes In the optimisation schemes, one can choose the weights of a network that should be updated and how and when to update those weights (see Section (5.1)).
When updating the weights, we can either tune or not tune the pre-trained weights:

- Do not change the pre-trained weights (feature extraction): One can train a linear classifier on top of the pre-trained representations. The best performance is typically achieved by using the representation not just of the top layer, but learning a linear combination of layer representations. For instance Peters et al. [2018] showed that exposing the deep internals of the pre-trained network is crucial, allowing downstream models to mix different types of semi-supervision signals. Ruder et al. [2018] presented an approach that learns a latent multi-task architecture that jointly addresses finding (a) the layers or subspaces that benefit from sharing, (b) the appropriate amount of sharing, and (c) the appropriate relative weights of the different task losses.

- Change the pre-trained weights (fine-tuning): The pre-trained weights are used as initialization for parameters of the downstream model. The whole pre-trained architecture is then trained during the adaptation phase.

The main motivation for choosing the order and how to update the weights is that we want to avoid overwriting useful pre-trained information and maximize positive transfer. Hence, practitioners choose to update the weights (1) progressively from top-to-bottom in time, (2) in intensity, or (3) compared to a pre-trained model:

1. Progressively in time (freezing): Training all layers at the same time on data of a different distribution and task may lead to instability and poor solutions. Hence, one should train layers individually to give them time to adapt to the new task and data (see Hinton et al. [2006b]). One can also select the combinations of layers that are trained together, and train all parameters jointly in the end (see the Universal Language Model Fine-tuning (ULMFiT) by Howard et al. [2018]).

2. Progressively in intensity (lower learning rates): To avoid overwriting useful information, we need to use lower learning rates. They are particularly important in lower layers (as they capture more general information), early in training (as the model still needs to adapt to the target distribution), and late in training (when the model is close to convergence). For instance, Howard et al. [2018]) used discriminative fine-tuning, which decays the learning rate for each layer. Liu et al. [2020] proposed a triangular learning rate schedule (also known as learning rate warm-up in Transformers) to maintain lower learning rates early in training.

3. Progressively versus a pre-trained model (regularisation): One way to minimize catastrophic forgetting is to encourage target model parameters to stay close to the parameters of the pre-trained model using a regularisation term (see Kirkpatrick et al. [2017]).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.5.5.3.3 Practical considerations Feature extraction requires adding more parameters than fine-tuning, making it slower to train a model (see Peters et al. [2019]). However, feature extraction is more space-efficient when a model needs to be adapted to many tasks, as it only requires storing one copy of the pre-trained model in memory. Adapters strike a balance by adding a small number of additional parameters per task. When it comes to performance, no adaptation method is clearly superior in every setting. If source and target tasks are dissimilar, feature extraction seems to be preferable. Otherwise, feature extraction and fine-tuning often perform similar, though this depends on the budget available for hyper-parameter tuning. Note, Transformers are easier to fine-tune (less sensitive to hyper-parameters) than LSTMs and may achieve better performance with fine-tuning. However, large pre-trained models (such as BERT-Large) are prone to degenerate performance when fine-tuned on tasks with small training sets. In practice, the observed behaviour is often on or off: the model either works very well or does not work at all as can be seen in the figure below. Understanding the conditions and causes of this behaviour is an open research question.

1.5.5.3.4 Getting more signal Since the target task (domain) is often a low-resource task, we can often improve the performance of transfer learning by combining a diverse set of signals (source setting) (see taxonomy in Figure (1.10)):

- Sequential adaptation: If some related tasks are available, one can fine-tune the model first on a related task with more data before fine-tuning it on the target task. This helps particularly for tasks with limited data and similar tasks (see Phang et al. [2019]), and improves sample efficiency on the target task (see Yogatama et al. [2019]).
- Multi-task fine-tuning: One can also fine-tune the model jointly on related tasks together with the target task. The related task can also be an unsupervised auxiliary task. Language modelling is a good choice for this and has been shown to help even without pre-training. For instance, Rei [2017] proposed a sequence labelling framework with a secondary training objective, learning to predict surrounding words for every word in the dataset. He showed that this language modelling objective incentivises the system to learn general-purpose patterns of semantic and syntactic composition. Language model fine-tuning is used as a separate step in ULMFiT. Recently, some improvements were obtained by using multi-task fine-tuning even with many target tasks. Liu et al. [2019c] presented a Multi-Task Deep Neural Network (MT-DNN) for learning representations across multiple tasks. It incorporates a pre-trained bidirectional transformer language model (BERT). Wang et al. [2019b] conducted a large-scale systematic study of candidate pretraining obtaining mixed results across pretraining tasks. Only modest gains from multitask training were observed.
- Dataset slicing: Rather than fine-tuning with auxiliary tasks, one can use auxiliary heads that are trained only on particular subsets of the data. To this end, one first analyse the errors of the model, use heuristics to automatically identify challenging subsets of the training data, and then train auxiliary heads jointly with main head.
- Ensembling: In order to improve performance of the predictions of models fine-tuned with different hyper-parameters, one can fine-tuned them with different pretrained models, or trained on different target tasks, or dataset splits may be combined.

1.5.6 Sentence similarity search

1.5.6.1 Description

As part of sentence processing, semantic matching aims at modelling the underlying semantic similarity or dissimilarity among different textual elements such as sentences and documents. Some applications include information extraction, top-k re-ranking in machine translation, question-answering, automatic text summarisation. We saw in Section (1.5.1) that natural language demonstrated complicated hierarchical structures, where different words could be organised in different orders to express the same idea. As a result, appropriate semantic representation of text plays a critical role in matching natural language sentences. We saw in Section (1.5.4) that representing text objects as bag-of-words (BoW), term frequency inverse document frequency (TF-IDF) vectors, or their enhanced variants, could not

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

accurately capture the similarity between individual words, and could not take the semantic structure of language into consideration. Hence, semantic search developed as an information retrieval system focusing on the meaning of the sentences rather than the conventional keyword matching. This technology is being leveraged to find similar sentences between two documents, or a key piece of information in a quarterly earnings report.
The main idea being to embed all entries in the corpus, which can be sentences, paragraphs, or documents, into a vector space. At search time, the query is embedded into the same vector space and the closest embedding from the corpus are found. These entries should have a high semantic overlap with the query. This is illustrated in Figure (1.12).

Figure 1.12: Semantic search.

For small corpora (up to about 1M entries) we can compute the cosine-similarity between the query and all entries in the corpus. However, for large corpora, sorting all scores is too time consuming. One solution is to use torch.topk to only get the top k entries. We distinguish two types of semantic search:

1. Symmetric semantic search: The query and the entries in the corpus are of about the same length and have the same amount of content. That is, one could potentially flip the query and the entries in your corpus. Some suitable models include:
    - paraphrase-distilroberta-base-v1 / paraphrase-xlm-r-multilingual-v1
    - quora-distilbert-base / quora-distilbert-multilingual
    - distiluse-base-multilingual-cased-v2
2. Asymmetric semantic search: The user has a short query (like a question or some keywords) and want to find a longer paragraph answering the query. In that setting, flipping the query and the entries in the corpus does not make sense. Some suitable models include:
    - msmarco-distilbert-base-v2

1.5.6.2 Some models

Depending on whether training data is available, prior research has proposed both unsupervised distance-based schemes and supervised deep learning schemes for sentence matching. For example, in the case of unsupervised learning, the Word Mover's Distance (WMD) metric (see Kusner et al. [2015]) measures the dissimilarity between two sentences

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(or documents) as the minimum distance to transport the embedded words of one sentence to those of another sentence. However, the sequential and structural nature of sentences is omitted in WMD. In the case of supervised learning, schemes based on deep neural networks have been proposed for sentence matching with the common characteristic that they adopt a Siamese architecture, taking the word embedding sequences of a pair of sentences (or documents) as the input, transforming them into intermediate contextual representations via either convolutional or recurrent neural networks, and performing scoring over the contextual representations to yield final matching results. However, many obvious compositional and hierarchical features are often either overlooked or not explicitly utilised. Some examples of Siamese networks based on CNN and BiLSTM are Mueller et al. [2016], Shao [2017], Wang et al. [2017]. Another approach is to use Transformer models, which we are now going to discuss.

As discussed above, in order to perform semantic search, we need to compute the embeddings for both the corpus and the query. One solution is to use Transformer models (see Section (14.4)) (they learn the contextual meaning of words and the relations between them), such as BERT (see Devlin et al. [2018]), RoBERTa (see Liu et al. [2019d]), for generating the sentence and text embeddings. BERT is self-supervised, meaning that it takes unlabelled data and automatically generates labels that it can train on. Effectively, some researchers started inputting individual sentences into BERT and deriving fixed-size sentence embeddings by averaging the BERT output layer. In those models, the semantic textual similarity is considered as a regression task so that the two sentences must be passed together (simultaneously) into the model when calculating the similarity score. As a result, they poorly scale for a real-life use cases, causing a massive computational overhead. For example, finding the most similar pair in a collection of 10k sentences requires about 50 million inference computations (about 65 hours on a V100 GPU) with BERT (see Reimers et al. [2019]). This is because it is not possible to compute the embeddings separately and calculate just the cosine similarity. In addition, the maximum sequence length (the total number of words/tokens the model can take at one pass) is shared between two documents, which causes the representations to be diluted due to chunking.

A better solution is to rely on sentence transformers (ST) as sentence embeddings, which are a set of models giving dense vector representations of sentences or paragraphs (see Section (1.5.4)). While they use Transformer models for generating the sentence and text embeddings, they are fine-tuned specifically for the task of semantic textual similarity (STS). Practically, one can use tools that index embeddings along the documents such as ElasticSearch and FAISS combined with SentenceBERT (SBERT) (see Reimers et al. [2019]) to perform semantic search. ElasticSearch gives the possibility of indexing dense vectors and using it for document scoring. FAISS is a library for efficient similarity search and clustering of dense vectors. It is a C++ based library built by Facebook AI with a complete wrapper in python that index vectorised data and perform efficient searches on them. SBERT uses siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity (see Section (14.5.2)). These modified and fine-tuned BERT NLP models are quite good at identifying similar sentences. This is illustrated in Figure (1.13).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Model	STS benchmark
Avg. GloVe embeddings	58.02
BERT-as-a-service avg. embeddings	46.35
BERT-as-a-service CLS-vector	16.50
InferSent - GloVe	68.03
Universal Sentence Encoder	74.92
Sentence Transformer Models	
bert-base-nli-mean-tokens	77.12
bert-large-nli-mean-tokens	79.19
bert-base-nli-stsb-mean-tokens	85.14
bert-large-nli-stsb-mean-tokens	85.29
roberta-base-nli-stsb-mean-tokens	85.44
roberta-large-nli-stsb-mean-tokens	86.39
distilbert-base-nli-stsb-mean-tokens	85.16

Figure 1.13: STS benchmark.

1.5.6.3 Examples

SBERT works very well for the tasks it was trained on, that is, comparing the semantic similarity on sentence level. However, if the number of sentences gets larger, or, the sentence length increases, usually the quality of the embedding vectors will suffer. The authors suggested splitting documents into passages if it has more than 512 tokens. We are now going to describe a few use cases of SBERT in real life applications.

As a first example, we consider a question-answer task where we want to find sentences that are similar to our search query. That is, find the most similar Quora question from our dataset to the query. To do so, we split the query into sentences and encode each of these using the SBERT model. Then, each of these sentence embeddings will be compared to every sentence in the previously embedded dataset using the cosine distance metric. The closer two embeddings are, the more similar their sentences should be. Thus, we can then rank the search results by using this metric.

In the case of information retrieval, hedge funds are leveraging semantic search to parse through and surface metrics within Quarterly reports (10-Q/10-K) and have them available as a quantitative trade signal in an instant once they are published. Here, we want to use SBERT to develop a recommender system based on the similarity of headlines. We give below two examples:

- Query (original headline): *European shares tread water ahead of Fed outcome*
- Output: *Sterling treads water ahead of UK budget*
  European shares steady ahead of Fed decision on rates
  UPDATE 1-Financials support FTSE as markets tread water ahead of Fed
  European stocks tread water as miners cap gains
  FOREX-Euro steady, supported by hopes of ECB action

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

European stocks tread water, all eyes on Fed
European shares steady after Fed, ahead of ECB
European shares ease ahead of Fed rate decision
EURO GOVT-Bunds steady, ECB liquidity anticipated
European shares steady, with ECB in focus

and

- Query (original headline): *Daimler awards 1,000 euro 'corona bonus' to German employees*
- Output: *VW to pay increased 4,100-euro bonus to German staff: document*
  Italy's Generali to raise up to 1.9 billion euros in German unit sale
  Germany's Volkswagen pays employees 4,750 euros bonus for 2018
  Germany to earmark 1 billion euros for EADS stake: sources
  German taxpayers to fund EnBW's USD 1 bln stock sale
  Germany's Benckiser to buy Peet's Coffee for USD 1 billion
  ING sells German shopping centre for 100 mln euros
  BRIEF-Ascom Wins Contract At About 1 Million Francs For German Power Plant
  Daimler to spend 100 mln euros to expand German battery output
  Volkswagen offers 830 million euros to compensate German buyers
  UPDATE 1-German biotech Evotec to buy Renovis for USD 152 mln
  EU authorities plan fines of up to 1 billion euro against German carmakers: Spiegel GSK to take 10 percent stake in Germany's CureVac for USD 163 mln
  Autogrill wins 150 mln euro contract in Germany

1.5.6.4 Extending the model

The SBERT is a bi-encoder model, as it encodes each sentence independently and map them to a dense vector space. As a result, it usually produces a lower performance in comparison with the BERT cross-encoder. This is because the BERT cross-encoder can compare both inputs simultaneously, while the SBERT bi-encoder has to map inputs independently to a meaningful vector space. This requires a large amount of training examples for fine-tuning the model. To remedy this drawback, Thakur et al. [2020] proposed the Augmented SBERT (AugSBERT), where they used the cross-encoder to label a larger set of input pairs (new input pairs) in order to augment the training data for the bi-encoder. The SBERT bi-encoder is then fine-tuned on this larger augmented training set (improving accuracy). Note, the authors are currently working on a project that evaluates multiple dense retrievers (SBERT, trained on MSMARCO dataset) and other lexical techniques which evaluate retrieval for a query over the large unlabelled dataset.

1.5.6.5 Transfer learning

While SBERT works very well for sentence similarity search, results are not good on downstream tasks such as event detection and topic recognition mechanism. As pointed from the authors, the purpose of SBERT sentence embeddings is not to be used for transfer learning on other tasks (see Section (1.5.5)). That is, in most of the cases you can not just add some layers on top of the sentence embeddings and hope that the latent space will align nicely to reflect the domain specific information.
To illustrate this point we consider document clustering and identify two cases:

1. In the case where the target document information is fine grained and the contained information therein relates to multiple abstract topics, one may need to jump through some loops in order to get meaningful document embeddings. This is also tightly related to the length of the target document. If we have long documents (which is usually correlated with the information contained in the documents), we will have to aggregate multiple embedded sentences, which can become very tedious with SBERT alone.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. In the case where the information contained in the document can be summarised in $1-3$ sentences, then SBERT embeddings can produce good results for document clustering.

1.6 The learning process

We briefly present the learning process, which we illustrate with a financial example consisting in forecasting the stock market. The reason being that it can be solved either as a regression problem or a classification problem. We let the underlying return be the stochastic process $x(t)$ at time t. The aim is to predict either its future value $x(t+1)$ or its future direction $\{-1, +1\}$.

1.6.1 Overview

1.6.1.1 Classification and prediction tasks

The strength of neural networks (NNs) compared to other techniques is their high capacity for classification, prediction and failure tolerance. For instance, in financial markets, NNs are generally applied for prediction tasks as well as for classification (see Section (3.3)). Classification and prediction tasks can be summarised as follows:

1+ Chaotic series prediction: performing time series forecast.
In the academic literature, chaotic systems are generally described by Mackey-Glass equations (see Mackey et al. [1977]). It is generated from the following non-linear time delay differential equation:

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x^n(t-\tau)} - bx(t)$$

where a, b, n and τ are real numbers. τ is the delay parameter. Depending on the parameter set we choose, the Mackey-Glass signal exhibits various characteristics. Figure (1.14) below demonstrates a particular behaviour, which is at the same time very chaotic. Similarly to a financial time series we observe: an identified behaviour (trending or mean-reverting) together with an unidentified one (chaos). Examples are next period price change and next period actual price (opening, high, low and closing prices), which concerns the change in price with respect to the next period and the numerical value for the actual next period price, respectively.

2+ Autonomous pattern generation: classification problem.
A pattern is a short sequence of randomly chosen real numbers that is repeated periodically to form an infinite time series. In general, the length of the sequence may be set from 1 up to the number of neurons N. In finance patterns are distinctive formations of market prices. For example, the next period direction concerns the direction of the price movement with respect to the next period. In that case we classify each record as Up or Down. The results are generally obtained by using only historic prices and technical indicators (TIs), which are mathematical transformations of market returns. The most common TIs are the simple moving average (SMA), exponential moving average (EMA), relative strength index (RSI), rate of change (ROC), moving average convergence / divergence (MACD), William's oscillator and average true range (ATR).

In general, academics and practitioners focus on case $(1+)$, which implies making a statement about the true dynamics (denoted f) followed by market returns. While in case $(2+)$ we only consider part of the probability space, this approach seems ignored by academics. From Computational Learning, an approximation model (the hypothesis, denoted h) must be devised and trained on the full measurement space (case $(1+)$) or part of the space (case $(2+)$). In order to assess if the function h is approximately correct, a measure of error must be considered (the cost function). This process requires sophisticated search and examination methods (optimisation algorithms) such as machine learning.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 1.14: Mackey-Glass Behaviour for (a, b, n, τ, x_0) = (0.2, 0.01, 10, 17, $random$)

1.6.1.2 Datasets

In order to test the accuracy of regression models, computer vision models, and NLP models, several challenging problems have been devised. For instance, the MNIST dataset [10] is considered as a hard dataset.

1.6.1.2.1 Computer vision In computer vision, the ImageNet Large Scale Visual Recognition Challenge is a benchmark in object classification and detection, with millions of images and hundreds of object classes. See Russakovsky et al. [2014] for some indication on the performance of convolutional neural networks on the ImageNet tests. COCO2017 is another dataset which has around 118k images.

1.6.1.2.2 NLP In order to test the accuracy of NLP models, several challenging NLP problems have been devised. Some examples include BLEU on the WMT 2014 English-to-German translation task, common sense reasoning (Stories Cloze Test), balanced classification (SNLI, version 1.1, see Bowman [2015]) question answering (SQuAD v1.1, see Rajpurkar et al. [2016]) (RACE, see Lai et al. [2017]), textual entailment (Multi-NLI, see Williams et al. [2018]), GLUE multi-task benchmark (see Wang et al. [2018b]), the STS benchmark (STSb) (see Cer et al. [2017]) provides a popular dataset to evaluate supervised STS systems.
The Stanford Natural Language Inference (SNLI) corpus is a collection of 570k human-written English sentence pairs manually labelled for balanced classification with the labels 'entailment', 'contradiction', and 'neutral', supporting the task of natural language inference (NLI), also known as recognising textual entailment (RTE). We give a few example pairs from the SNLI corpus in Table (1.1). Each has the judgements of five mechanical turk workers and a consensus judgement.

Text	Judgement	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction CCCCC	The man is sleeping.
An older and younger man smiling.	neutral NNENN	Two men are smiling and laughing at the cats playing on the floor
A soccer game with multiple males playing.	entailment EEEEE	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	neutral NNECN	A happy woman in a fairy costume holds an umbrella.

Table 1.1: Example pairs from the SNLI corpus

[10] *http://yann.lecun.com/exdb/mnist/*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The Multi-NLI corpus contains 430k sentence pairs and covers a range of genres of spoken and written text. In the STS benchmark (STSb), the data includes $8,628$ sentence pairs from the three categories captions, news, and forums. It is divided into train $(5,749)$, dev $(1,500)$ and test $(1,379)$.
Note, Dor et al. [2018] used Wikipedia to create a thematically fine-grained train, dev and test set for sentence embeddings methods. The Wikipedia articles are separated into distinct sections focusing on certain aspects. They assumed that sentences in the same section were thematically closer than sentences in different sections and created a large dataset of weakly labelled sentence triplets: The anchor and the positive example come from the same section, while the negative example comes from a different section of the same article. For example, from the Alice Arnold article: Anchor: **Arnold joined the BBC Radio Drama Company in 1988., positive: Arnold gained media attention in May 2012., negative: Balding and Arnold are keen amateur golfers**.

1.6.2 Training and exploitation of the model

When forecasting time series or, generating autonomous patterns, we consider the following three steps:

1. Training: it consists in determining the model parameters that minimise the cost function.
2. Optimisation: identify the best model in the model space.
3. Adaptability: generalisation assessment.

The optimisation of the model is performed in the validation set. The data in the testing set is the sample where the calibrated model is assessed for generalisation. That is, the model is ran without any exhaustive filter nor complementary optimisation.

1.6.2.1 Sample hashing

The model will improve based on the loss generated by the data. However, the data is a sample, which means that there is always a difference between the sample distribution and the population distribution. To avoid overfitting, we split the available data into three samples: Training, Validation and Testing. It is called sample hashing or simple hold-out splits. Once we have selected a model that performs well on the training data, we run the model on validation dataset. It helps provide an unbiased evaluation of the model's fitness. If the error on the validation dataset increases then we have an overfitting model. The test dataset contains data that has never been used in the training. It helps with final model evaluation. This is illustrated with an example in Figure (1.15).

Figure 1.15: Sample hashing: Training sample is $washed - out$ at 20% meaning that the Ridge regression is only performed on the 80% remaining. By doing this, we get rid of the reservoir initial transient.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Even though this approach is inexpensive in terms of computational power, it is also a very simplistic approach. Some of the problems are as follows (see Chollet [2017]):

- Data representativeness: all datasets must represent the patterns in the population as much as possible. This becomes especially important when you generate samples from a sample. Random shuffling may help solve this problem.
- The arrow of time: when the dataset is a time series, it is likely ordered chronologically. Shuffling randomly and then performing simple hold-out validation no-longer make sense.
- Data redundancy: if some samples appear more than once, a simple hold-out split with random shuffling may introduce redundancy between training and testing datasets. That is, identical samples belong to both datasets, which is problematic as data used for training thus leaks into the dataset for testing implicitly.

1.6.2.2 Cross-validation

Several techniques exist for building and testing a model when the available data is limited. Cross-validation (CV) is a statistical method used to compare and select a model for a given predictive modelling problem (see Abu-Mostafa et al. [2012]). It is a re-sampling procedure used to evaluate machine learning models on a limited data sample. It has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
   (a) Take the group as a hold out or test data set
   (b) Take the remaining groups as a training data set
   (c) Fit a model on the training set and evaluate it on the test set
   (d) Retain the evaluation score and discard the model
4. Summarise the skill of the model using the sample of model evaluation scores

Each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model $k-1$ times. This is illustrated in Figure (1.16).

Figure 1.16: Cross Validation Diagram.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

It is important that any preparation of the data prior to fitting the model occur on the CV-assigned training dataset within the loop rather than on the broader data set. A failure to perform these operations within the loop may result in data leakage and an optimistic estimate of the model skill (see Russell et al. [2009]). The results of a k-fold cross-validation run are often summarised with the mean of the model skill scores. It is good practice to include a measure of the variance of the skill scores, such as the standard deviation or standard error. A poorly chosen value for k may result in a mis-representative idea of the skill of the model, such as a score with a high variance (that may change a lot based on the data used to fit the model), or a high bias, (such as an overestimate of the skill of the model) (see Section (10.1.1.3)). Three common tactics for choosing a value for k are as follows (see Kuhn et al. [2018]):

- Representative: The value for k is chosen such that each train/test group of data samples is large enough to be statistically representative of the broader dataset.
- $k = 10$: The value for k is fixed to 10, a value that has been found through experimentation to generally result in a model skill estimate with low bias a modest variance.
- $k = n$: The value for k is fixed to n, where n is the size of the dataset to give each test sample an opportunity to be used in the hold out dataset. This approach is called leave-one-out cross-validation.

1.6.2.3 Variations to cross-validation

There exists a number of variations to the k-fold cross validation procedure. Some commonly used variations are as follows:

- Train/Test Split: Taken to one extreme, k may be set to 2 (not 1) such that a single train/test split is created to evaluate the model.
- LOOCV: Taken to another extreme, k may be set to the total number of observations in the dataset such that each observation is given a chance to be the held out of the dataset. This is called leave-one-out cross-validation, or LOOCV for short. This approach is far less bias since we use the entire dataset for training compared to the validation set approach where we use only a subset of the data for training. There is no randomness in the training/test data since performing LOOCV multiple times yields same results. For example, given a data set with n observations, the training data contains $n - 1$ observations, and test data contains 1 observation. This process is iterated n times for each data point, generating n Mean Square Error(MSE) (see Khandelwal [2019]). This is illustrated in Figure (1.17).

Figure 1.17: LOOCV cross validation.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Stratified: The splitting of data into folds may be governed by criteria such as ensuring that each fold has the same proportion of observations with a given categorical value, such as the class outcome value. This is called stratified cross-validation. It forces each fold to have at least m instances of each class and it ensures that one class of data is not overrepresented especially when the target variable is unbalanced. For example, consider a binary classification problem where we want to predict if a passenger on Titanic survived or not. We ensure that each fold has a percentage of passengers that survived and a percentage of passengers that did not survive. This is illustrated in Figure (1.18).

Figure 1.18: Stratified cross validation.

- Repeated: This is where the k-fold cross-validation procedure is repeated n times, where importantly, the data sample is shuffled prior to each repetition, which results in a different split of the sample.

- Time-series Cross Validation: This is a modification to the cross validation in the case where the dataset is a time series. We use forward chaining, also referred as rolling-origin, where the origin at which the forecast is based rolls forward in time. For example, in a daily forecast, each day is a test data and we let the data from the previous days be the training set. We start training the model with a minimum number of observations and use the data for the next day to test the model. We then keep moving through the data set. It ensures that we consider the time series aspect of the data for prediction (see Khandelwal [2019] proposed). This is illustrated in Figure (1.19).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 1.19: Time series cross validation. $D1, D2, D3, ...$ is is a series of days. The days highlighted in blue are used for training and the days highlighted in yellow are used for testing.

1.6.2.4 Bootstrapping

Bootstrapping is another technique for dealing with limited data. Given a small dataset of size n, a bootstrap sample of size n is produced from the original dataset by randomly selecting n data items from it with replacement. Any number of bootstrap samples of size n can be produced by repeating this process. However, some data items may be present multiple times in the bootstrap sample.

1.6.3 Financial example

We illustrate the learning process with a financial example consisting in forecasting the stock market.

1.6.3.1 The framework

1.6.3.1.1 The input variables When selecting the right input variables, Krollner et al. [2010] showed that academics and practitioners rely on some form on lagged index data (case $(1+)$ in Section (1.6)) rather than considering autonomous pattern generation (case $(2+)$). Examples of articles that focus on forecasting market returns are Zeng et al. [2006], Zhang et al. [2007], Zhu et al. [2008], Lu et al. [2009], Liao et al. [2010]. Based on some well known techniques combining forecasts (see Armstrong [2001]), several studies showed that combining forecasts (using NN models) could greatly enhance the forecast effectiveness (see Wang et al. [2008], Chitra et al. [2010]).
Alternatively, traders and academics have used technical analysis as well as certain fundamental valuation metrics to improve results (see Gencay [1997], Metghalchi et al. [2007]). Further, some statisticians showed that technical indicators have predictive power and found a good measure of support for their effectiveness (see Chen et al. [2009], Metghalchi et al. [2012] [2015]). As a result, these strategies have been incorporated in machine learning models to forecast future trend direction (case $(2+)$). The results obtained are achieved using only historic prices and technical indicators (see Kim rt al. [2006], Bekiros et al. [2008], Kara et al. [2011], Dingli et al. [2017]). Some authors also used technical indicators to forecast both future price levels and asset price directions (see Patel et al. [2015]). Note, one can directly use continuous technical indicators (actual time series) or one can choose to represent them as trend deterministic data (discrete in nature). Van den Poel et al. [2016] combined in some way continuous technical indicators, technical trading rules and past returns to become input to predictor models. They proposed a predictive model to predict future stock price direction accurately.

1.6.3.1.2 The data The different forecasting outputs are computed for the following periods: daily, weekly, monthly, quarterly and yearly. However, an increasing number of articles consider intra-day high frequency data, which requires highly liquid markets. In that case the data comprises the date and time, open-high-low-close price levels and the volume traded for each underlying asset. One of the main reason for considering intraday trading horizon is the notion of self-destruction of predictable patterns in stock prices (see Timmermann et al. [2004]). For the data to be

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

consistent, historical prices and/or volume are usually adjusted to reflect spin-offs, stock splits/consolidations, stock dividend/bonus and right offerings/entitlement.

1.6.3.1.3 Evaluation methods Most evaluation methods of the ML models are benchmark models where the proposed improved version is compared against the original one using a forecast error as an evaluation metric. Examples of benchmark models are buy and hold, random walk, statistical techniques and other machine learning techniques.

1.6.3.1.4 The measures Linear measures are measurements in one direction. For example length, width, or height of objects, and distance between two points are linear measures. Linearity also also express the idea that the model possesses the property of additivity and homogeneity, meaning that a change in one variable causes a proportional change in another variable. An example of a linear measure is the root-mean-square deviation (RMSD) or root-mean-square error (RMSE), which is used to measure the differences between values predicted by a model or an estimator and the values observed.
In mathematics and science, a nonlinear system is a system in which the change of the output is not proportional to the change of the input. Systems can be defined as nonlinear, regardless of whether known linear functions appear in the equations. Nonlinearity measures are a means of quantifying the size of nonlinearity in the Input/Output-behaviour of nonlinear systems (more than one dimension). Methods that efficiently compute nonlinearity measures are based on convex optimisation.
Note, linear measures do not account for the non-linearity of RNN models (see Bellgard et al. [1999]). One way around is to assess the performance of a model using trading simulation.

1.6.3.2 Using neural networks to model returns

1.6.3.2.1 An overview Among the various techniques to classifying and forecasting financial time series, fundamental and technical analysis are the most popular ones. Even though statistical procedures are widely used for patterns recognition, the effectiveness of these methods relies both on model's assumptions and prior knowledge on data properties. To remedy these pitfalls, several classifiers developed, using various data mining and computational intelligence methods such as rule induction, fuzzy rule induction, decision trees, neural networks etc. In 1990, Kimoto et al. [1990] applied a modular neural network machine learning algorithm to predict the movement of stock index of Tokyo Stock Exchange. Since ANN developed further, it has been widely applied to stock analysis. Nowadays, the best recognised tools in the currency markets is the artificial neural networks (ANNs), supported by numerous empirical studies (see Ahmed et al. [2010]). Similarly, Krollner et al. [2010] provided a survey on forecasting financial time series with machine learning and found that ANNs were the dominant technique in that field. The foremost reason for using ANNs is that there is some nonlinear aspect to the forecasting problem under consideration, taking the form of a complex nonlinear relationship between the independent and dependent variables. The characteristics of financial time series, such as equity stock or currency markets, are influenced by the psychology of traders (behavioural finance) and are strongly non-linear and hardly predictable (see Maknickiene et al. [2011]).

One advantage in using an ANNs is that the researcher does not need to know a priori the type of functional relationship existing between the independent and dependent variables (see Darbellay et al. [2000]). A vast literature demonstrated that neural network performs better than conventional statistic approaches in financial forecasting (see Refenes et al. [1994], Adya et al. [1998], Abu-Mostafa et al. [2001]). For many financial forecasting problems, classification models work better than point prediction. Further, Qi [2001] argued that due to the continually changing nature of financial relationships, ANNs are more likely to outperform traditional techniques when the input data is kept as current as possible. This is done by recursive modeling, where the researcher adds new observations and drops the oldest ones each time a new time series forecast is made (sliding window). Olson et al. [2003] compared neural network forecasts of one-year ahead Canadian stock returns with the forecasts obtained by using ordinary least squares (OLS) and logistic regression techniques and showed that the backpropagation algorithm outperformed the best regression alternatives for both point estimation and in classifying firms expected to have either high or low returns This superiority of the NNs translated into greater profitability using various trading rules. Using data from

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

four major stock market indexes, Fok et al. [2008] compared linear regression and neural network backpropagation by testing their forecasting performances. They showed that the latter had better prediction accuracy.

1.6.3.2.2 Combining neural networks with evolutionary algorithms Some of the disadvantages of NNs are long training time, unwanted convergence to local instead of global optimal solution, and large number of parameters (see Lee et al. [1991]). One way forward is to combine ANN with another algorithm such as evolutionary computing tools that can take care of a specific problem (see Section (20.2)). Many research papers have appeared in the literature using evolutionary computing tools, such as genetic algorithm (GA) (see Montana et al. [1989], Nair et al. [2011], Contras et al. [2016]), particle swarm optimization (PSO) (see Kennedy et al. [1995] Jordehi et al. [2013]), bacterial foraging optimization (BFO) (see Jhankal et al. [2011]), and Adaptive bacterial foraging optimization (ABFO) in developing forecasting models.
Inspired by the pattern exhibited by bacterial foraging behaviour, Helstrom et al. [1998] proposed an evolutionary computation technique called Bacterial foraging optimization (BFO). It analyses how the run-length unit parameter of BFO controls the exploration of the whole search space and the exploitation of the promising areas. Shin et al. [1998] proposed data mining approach using genetic algorithms (GA) to solve the knowledge acquisition problems that are inherent in constructing and maintaining rule-based applications for stock market. Kim et al. [2000] used GA to improve the learning algorithm, and also to reduce the complexity of the feature space. Kim et al. [2004b] developed a feature transformation method using genetic algorithms. It reduces the dimensionality of the feature space and removes irrelevant factors involved in stock price prediction. Jamous et al. [2016] considered particle swarm optimization with center mass (PSOCOM) technique to develop an efficient forecasting model. Rani et al. [2017] presented an overview on optimisation techniques and hybridisation to improve ANN performance (see details in Section (20.2)).

1.6.3.3 The learning process

When forecasting the stock market, the learning process is as follows:

- In the case of pattern generation: During the training phase, the reservoir computer receives the pattern signal as input and is trained to predict the next value of the pattern from the current one. The length of the training input sequence is measured in terms of the pattern length, that is, how many times the whole pattern is presented to the reservoir computer. The duration of the training process depends on the length of the pattern. For instance, short patterns are successfully learnt after 100 repetitions, while long patterns need to be shown up to $50k$ times. The RMSE is used to evaluate the training phase. After the training phase, the reservoir input is switched from the training sequence to the reservoir output signal, and the system is left running autonomously.
- In the case of series prediction, training and test phases are the same as for pattern generation. During training, the reservoir receives the time series and is trained to perform a one-step ahead prediction, while in the test phase the reservoir receives its own output and the system runs autonomously.

Based on the classification task chosen, several possibilities exist for training and exploiting models:

1* training on transient process → exploitation on transient process

2* training on attractor-based model → exploitation on attractor-based

3* training on attractor-based model → exploitation on transient process

4* training on attractor-based model → exploitation on transient process

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.7 ML pipeline

1.7.1 Overview

We have introduced machine learning (ML) in Section (1.2) and presented a few models with different applications in Sections (1.3) (1.4) (1.5). We now need to describe the productionisation of ML and model management.
Building production machine learning applications requires to record experiments, ensure reproducible runs, and manage and deploy models. To do so, we need a standard platform where models generalise well and updates are performed appropriately. This is because if the environment changes, such as input data, and the models are not regularly assessed in the context of what they were built for, it will result in the models losing their relevance and values over time.

Data science and engineering can be comprehended as follows: The people feeding the data in are engineers, and the people designing the model are scientists.
Production environment: ML models are usually reused from existing research (transfer learning), so that the learning is public. Based on the data used, the models have biased and could be broken. Thus, one need to understand how models should live in an environment that continuously ingests new data, integrates new code, is called by end-users, and can fail in a variety of ways from time to time. One also need to understand the models and algorithms of machine learning. We need to implement the ML workflow such that it is reproducible and stable in time. The best solution is to automate that process and monitor it.

The ML Production Value Chain is a high-level description of what is required for deploying models to end-users. The steps are as follows:

- Collaboration data exploration: Need to establish a common language between engineers and scientists.
- Problem solving: Define the problem to be solved, perform exploratory data analytic, identify the ML models to test.
- Data integration:
  - Real-time data delivery to cloud (migration technology).
  - Data lake: AWS, Azure, Google Cloud Platform, Databricks, Kafka, among others.
  - Streaming architectures: data pipeline automation.
- Data management.
  - Data ingestion.
  - Data cleansing.
  - Data analysis.
  - Data transform: When applying machine learning to real world datasets, a lot of effort is required to preprocess data into a suitable format. This includes converting between formats, tokenising and stemming text and forming vocabularies, and performing a variety of numerical operations such as normalisation.
  - Data validation: understand, validate, and monitor data at scale.
- ML models development:
  - Train, validate, and test models.
  - Hyperparameter tuning: Select the optimal statistical model and the optimal feature extractor. This is done simultaneously with the training and validation process. This is usually done offline.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

  - Perform model analysis: Before deploying any machine learning (ML) model, ML developers need to evaluate model performance to ensure that it meets specific quality thresholds and behaves as expected for all relevant slices of data.
  - Identify and solve bias.
  - Speedup models post-training before going to production.

- Data pipeline management:
  - Manage pipeline complexity: The data pipeline that sits behind the ML models must be handled with care.
  - Manage versions: ML serving systems need to support model versioning (for model updates with a roll-back option) and multiple models (for experimentation via A/B testing), while ensuring that concurrent models achieve high throughput on hardware accelerators (GPUs and TPUs) with low latency.

- Model management:
  - Improve model quality: With changing environment ML models must be retrained to be accurate, relevant, sustainable, and responsible. Extract data to train, test, and validate models, while simultaneously optimising, governing, and reporting on the model. Improve model quality.

- Hosting model: models must be accessible at all times to end-users.

The Production Framework, presented below, provides a more detailed understanding. The data scientists are concerned with the ML model implementation phase, while the data engineers are concerned with the data pipeline as well as hosting and model management. The latter need to track model performance to ensure longevity, and deploy the models to a source where they can be accessed through an API by end-users. They must apply DevOps principles to enable reliability and accessibility.

The basic idea is to have a machine learning software (or platform) where we can serve the data to the machine, get metrics and results, then ship the results to clients. One can

1. build all of this custom: Google cloud / Aws covers the first step.
2. use pieces that are open source: Neptune, MLflow cover the second step.
3. use end-to-end solutions: Floydhub, PipelineAI.

Floydhub is used for data scientist that needs to check metrics, parameters, without relying on engineers. Google is used once you are ready to deploy the ML model. PipelineAI combines a lot of different modules including the orchestration of different steps of the pipeline or different models, if you have multiple ML models involved in he process. Covering all the steps is always good for prototyping, but when we need to scale the project, then either these solutions are restrictive or they are expensive.

1.7.2 Some platforms

Machine learning algorithms can learn from multiple integrated sources as well as from previous experience. When developing a machine learning project, if you have millions of training data, or, your algorithm is sophisticated and take a long time to execute, you need a machine learning software (or platform). It aims at developing a machine with these prominent specifications. Several model management tools developed to handle the archiving of ML artefacts, metrics and pickled model files. We are going to list some of these platforms (the list is not exhaustive):

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.7.2.1 Build all custom

- Google Cloud ML Engine: It is a hosted platform where developers and data scientists develop and run high-quality machine learning models and datasets. Insight of this ML and Artificial Intelligence Framework:
  - Provides AI and ML model building, training, predictive modelling, and deep learning.
  - The two services, namely training and prediction, can be used jointly or independently.
  - This software is used by the enterprises. For instance, detecting clouds in a satellite image, responding faster to customer emails.
  - It can be used to train a complex model.
- Amazon Machine Learning (AML): It is a robust and cloud-based machine learning and artificial intelligence software which can be used by all skill levels of developers. This managed service is used for building machine learning models and generating predictions. It integrates data from multiple sources: Amazon S3, Redshift, or RDS. Insight of this AI and Machine Learning Framework:
  - Amazon Machine Learning provides visualization tools and wizards.
  - Supports three types of models, that is, binary classification, multi-class classification, and regression.
  - Permits users to create a data source object from the MySQL database.
  - It permits users to create a data source object from data stored in Amazon Redshift.
  - Fundamental concepts are Data sources, ML models, Evaluations, Batch predictions, and Real-time predictions.

1.7.2.2 ML models libraries

- Accord.NET: It is a .Net machine learning framework combined with audio and image processing libraries written in $C\#$. It consists of multiple libraries for a wide range of applications, that is, statistical data processing, pattern recognition, and linear algebra. It includes the Accord.Math, Accord.Statistics, and Accord.MachineLearning. Insight of this Artificial Intelligence Framework:
  - Used for developing production-grade computer vision, computer audition, signal processing, and statistics applications.
  - Consists of more than 40 parametric and non-parametric estimation of statistical distributions.
  - Contains more than 35 hypothesis tests, including one way and two-way ANOVA tests, non-parametric tests like Kolmogorov-Smirnov test and many more. It has more than 38 kernel functions.
- Shogun: It is an open source machine learning library, Shogun, was first developed by Soeren Sonnenburg and Gunnar Raetsch in 1999. This tool is written in C^{++}. It provides data structures and algorithms for machine learning problems. It supports many languages like Python, R, Octave, Java, $C\#$, Ruby, Lua, etc. Insight of this Artificial Intelligence Framework:
  - This tool is designed for large scale learning.
  - It focuses on kernel machines like support vector machines for classification and regression problem.
  - Allows linking to other AI and machine learning libraries like LibSVM, LibLinear, SVMLight, LibOCAS, etc.
  - It provides interfaces for Python, Lua, Octave, Java, $C\#$, Ruby, MatLab, and R.
  - It can process a vast amount of data, like 10 million samples.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Apache Spark MLlib: It is a scalable machine learning library. It runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. Also, it can access data from multiple data sources. Several algorithms are included like for
  - Classification: logistic regression, naive Bayes,
  - Regression: generalized linear regression,
  - Clustering: K-means, and many more.

  Its workflow utilities are Feature transformations, ML Pipeline construction, ML persistence, etc. Insight of this AI and Machine Learning Framework:
  - Ease of use. It can be usable in Java, Scala, Python, and R.
  - MLlib fits into Spark's APIs and interoperates with NumPy in Python and R libraries.
  - Hadoop data source like HDFS, HBase, or local files can be used. So it is easy to plug into Hadoop workflows.
  - It contains high-quality algorithms and outperforms better than MapReduce.

1.7.2.3 ML models software

- Apache Mahout: It is a distributed linear algebra framework and mathematically expressive Scala DSL. It is a free and open source project of the Apache Software Foundation. The goal of this framework is to implement an algorithm quickly for data scientists, mathematicians, statisticians. Insight of this AI and Machine Learning Framework:
  - An extensible framework for building scalable algorithms.
  - Implementing machine learning techniques, including clustering, recommendation, and classification.
  - It includes matrix and vector libraries.
  - Run-on the top of Apache Hadoop using the MapReduce paradigm.
- TensorFlow: It is an open source machine learning library which helps you to develop your ML models. It was developed by the Google team (see Dean et al. [2015]). It has a flexible scheme of tools, libraries, and resources that allows researchers and developers to build and deploy machine learning applications. Insight of this AI and Machine Learning Framework:
  - An end-to-end deep learning system.
  - Build and train ML models effortlessly using intuitive high-level APIs like Keras with eager execution.
  - This open source software is highly flexible.
  - Performs numerical computations using data flow graphs.
  - Run-on multiple CPUs or GPUs (with optional CUDA and SYCL extensions), and also on mobile computing platforms.
  - Efficiently train and deploy the model in the cloud.
- TensorFlow Extended (TFX): It is an end-to-end platform for deploying production ML pipelines. It allows you to move your models from research to production. A TFX pipeline is a sequence of components that implement an ML pipeline which is specifically designed for scalable, high-performance machine learning tasks. That includes modelling, training, serving inference, and managing deployments to online, native mobile, and JavaScript targets. The pipeline components are built using TFX libraries which can also be used individually. Insight of this AI and Machine Learning Framework:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

  - TF Data validation: It helps developers understand, validate, and monitor their ML data at scale.
  - TF Transform: Preprocess and normalise data.
  - TF Model Analysis: It enables developers to compute and visualise evaluation metrics for their models.
  - TF Serving: Support model versioning and multiple models.

- Torch: It is a framework with maximum flexibility and speed to build your scientific algorithms. It provides support for artificial intelligence and machine learning algorithms. It is easy to use and efficient scripting language based on Lua programming language. It also provides a wide range of deep learning algorithms. Insight of this ML and AI Software:

  - Provides a powerful N-dimensional array that supports lots of routines for indexing, slicing, and transposing.
  - It has s splendid interface to C, via LuaJIT.
  - Fast and efficient GPU support.
  - This framework is embeddable with ports to iOS and Android backends.

- Neptune: It is a collaboration hub for data science / machine learning teams that focuses on three areas:

  1. Track: all metrics and outputs in your data science or machine learning project. It can be model training curves, visualisations, input data, calculated features and so on.
  2. Organise: automatically transform tracked data into a knowledge repository.
  3. Collaborate: share, compare and discuss your work across data science project.

  Insight of this AI and Machine Learning Framework:

  - Track and version your notebooks: Log all your notebooks directly from Jupyter or Jupyter Lab.
  - Manage your experimentation process: Neptune tracks your work with virtually no interference to the way you like to do it. Decide what is relevant to your project and start tracking: Metrics, Hyperparameters, Data versions, Model files, Images, Source code.
  - Integrate with your workflow easily: Neptune is a lightweight extension to your current workflow. Works with all common technologies in data science domain and integrates with other tools. It will take you 5 minutes to get started.
  - Compare notebooks like source code: Record your exploration process and analyse differences between checkpoints. Select two notebooks and compare their content, code and outputs, side-by-side just like source code.

- MLflow: It is an open source platform to manage the ML life cycle, including experimentation, reproducibility and deployment. It currently offers three components:

  1. Tracking: Record and query experiments: code, data, config, and results.
  2. Projects: Packaging format for reproducible runs on any platform.
  3. Models: General format for sending models to diverse deployment tools.

  The MLflow Tracking component is an API and UI for logging parameters, code versions, metrics, and output files when running your machine learning code and for later visualising the results. MLflow Tracking lets you log and query experiments using Python, REST, R API, and Java API APIs. An MLflow Model is a standard format for packaging machine learning models that can be used in a variety of downstream tools. For example, real-time serving through a REST API or batch inference on Apache Spark.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.7.2.4 End-to-end solutions

- FloydHub: It is a fast and easy way to build, train, and deploy deep learning models. Workspaces are fully configured development environments for deep learning on the cloud.
  - Launch a GPU-enabled workspace in 1-click.
    - Jupyter Notebooks and scripts
    - Built-in metrics and alerts
    - Github integration
  - Train at scale and collaborate. Use the command-line interface to run and monitor reproducible experiments and their results directly from any terminal.
    - Train in parallel
    - Full reproducibility
    - Integrates with your workflow
  - Deploy and scale instantly. Go from model to fully-scalable deployment with a single command. Within a few seconds, your model is available as an API.
    - Scale based on your needs
    - Share auto-generated web page
    - Integrate into your application.
  - Built for data scientists. Move the mundane tasks out of the way and provide tools so you can relentlessly focus on your AI.
  - Full reproducibility: End-to-end version control of your code, data and results.
    - Experiment management: Easily track and organize your work.
    - Massive concurrency: Run hyperparameter sweeps or work on multiple projects.
    - Data versioning: Automatic data versioning and management.
    - Training Metrics: Track progress and analyze results.
    - Notifications and Insights: Built-in smart alerts to save you time and money.
    - Integrations: Integrate with GitHub, Slack, S3 and more.
    - Effortless Automation: Automate your workflows with our CLI, API and webhooks.
  - Collaborate seamlessly: See your project's big picture at a glance. Check your team mates' progress, see latest results and share feedback, without interruption or opaqueness of local development.
    - Central hub to avoid duplicative work
    - Role-based permissions
    - Consolidated billing
    - Unlimited private projects
- Dataiku: It is an end-to-end platform based on open source that provides self-service analytics, and permits easy deployment of machine learning models in production. It combines all existing tools for data scientists into one place. Insight of this ML and AI Software:
  - Insights: It allows to easily connect to existing infrastructure and produces instant insights from data. Data can be uploaded from files, Hadoop, NoSQL, SQL, cloud storages. It allows for interactive data cleaning, data exploration, and enrichment features as well as integrated advanced processors. It generate graphs to visualise the data which can be shared and pushed to webapps.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Models: It offers the use of a visual user interface (UI) or code to leverage the latest machine learning technologies and fine tune these models to optimise predictions. As soon as the models are built and assessed, they are fully production ready.
- AI-enabled service: Deploy machine learning models to production in one click with Kubernetes, or, deploy as an API. It makes it smoother to get AI models operational (including maintenance and monitoring) and to integrate them with business processes.

Models use multiple engines for running ML tasks: MLLib is running on Spark, H2O which is using Sparkling Water (in memory engines). Spark acts as executor. One can also add custom models in MLLib. H2O holds deep learning models. Get comprehensive performance reports: confusion matrix, decision chart, lift charts, ROC curve, density chart, detailed metrics. Visualise important variables.
The production architecture is made of a design node and production nodes which consists of batch node and real-time node.

- Design environment: The design node is a place where both data scientists and business analyst collaborate. This is where we get the insights and build the ML models to perform predictive analytics. It is made of several features that can be rolled back. It is a place to:
  * Assemble code and visualise components.
  * Create workflows as a set of datasets and recipes (follow script made of steps). In a flow, one can use code recipes to execute a piece of user-defined code. Use notebooks to prototype and test code.
  * Define dependencies between datasets and use partitioning to manage incremental data.

  DSS will automatically manage storage formats, schemas, and data movement. We can then go from design to production.
  Create a package containing all the information on the model. Take the flow (datasets metadata and all dependencies (connections, table names, schema, file-path)) and create a bundle. The outflow of the design is a bundle that contains: the flow definition, optional persisted data, charts, insights, notebook. The bundle is then loaded to production environment. It is then versioned and can be rolled back.
- Production environment:
  * The batch node is a place where everything done in the design environment can be duplicated into an environment to be reran for model improvement. Create automation scenarios, including steps such as triggers, checks, notifications. Use GUI or Python API for advanced scenarios. One can use the scenarios to automatically execute recipes. Through a scheduler, the analysis are regularly reran with new changing data. The system can be monitored and new data checked.
  * Alternatively, we can use real-time node to directly query the ML model at any time to get a prediction.
  * API node: take automated environment (model and data from batch node) and make it available for real-time query. It provides an automatic plug-in showing how people are using the models.
  * API scoring node: In order to score an entity, it goes through a pipeline taking raw input and transforming it prior to scoring models: raw request features as input, enrich with features using lookups in production DBs, create derived features, the latter is used to score the model.
  * ML API: Automate all aspects the life cycle of ML models: visual analysis, tuning settings, training models, checking results, deploying saved models to flow and retraining them.
  * API deployer: It is a meta API node which keeps track of every API nodes constructed and all services at endpoints deployed across these nodes. Thus, one can create a bundle at the design level and turn it into several services at production level. The API services must have different keys even though they come from the same design. The nodes can do lookups in the API endpoints.
  * API endpoint: one API endpoint calling all the API endpoints.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- PipelineAI: It continuously trains, optimises, and serves machine-learning models on live-streaming data directly in production. The platform gives data scientists and engineers the freedom to quickly deploy, test, and rollback their models directly in production. It provides the tooling, infrastructure, and dashboards necessary to directly manage production. It supports all major AI and machine-learning frameworks, including Spark ML, Apache Kafka, Scikit-Learn, Xgboost, R, TensorFlow, Keras, and PyTorch. It is an end-to-end ML pipeline combining TFX, KubeFlow, Airflow, MLflow, and TPU,

1.7.3 Production framework

We have introduced computer cluster in Section (1.1.3.3) as a way of getting more computing power and better reliability by orchestrating a number of low-cost computers. This approach is used to handle big data expressed as multiple data sources as well as to perform complex computing tasks. We are going to consider computer cluster to describe a bespoke ML production framework that can stream external data sources (public data) as well as internal IoT (internet of things) system.

1.7.3.1 Terminology

1.7.3.1.1 Cluster manager We briefly present some multi-cloud clustering and orchestration technology (see Section (1.1.3.4)). Scientists can now run their big data stack (Spark, Presto, Kafka, etc.) on Kubernetes in a much simpler manner. Furthermore, they can run all of the post-Hadoop AI and data science tools like Jupyter, TensorFlow, PyTorch, or, custom Docker containers, on the same cluster.

- Apache Hadoop is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation. It provides a software framework for distributed storage and processing of big data using the MapReduce programming model (see Dean et al. [2004]). Originally designed for computer clusters built from commodity hardware, it has also found use on clusters of higher-end hardware. The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming model.
  The base Apache Hadoop framework is composed of the following modules:

  - Hadoop Common: Contains libraries and utilities needed by other Hadoop modules.
  - Hadoop Distributed File System (HDFS): A distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster.
  - Hadoop YARN: A platform responsible for managing computing resources in clusters and using them for scheduling users' applications.
  - Hadoop MapReduce: An implementation of the MapReduce programming model for large-scale data processing.

  The term Hadoop is often used for both base modules and sub-modules and also the ecosystem. By default Hadoop uses FIFO scheduling, and optionally 5 scheduling priorities to schedule jobs from a work queue. The capacity scheduler supports several features that are similar to those of the fair scheduler.

  1. Queues are allocated a fraction of the total resource capacity.
  2. Free resources are allocated to queues beyond their total capacity.
  3. Within a queue, a job with a high level of priority has access to the queue's resources.

- Originally designed by Google, Kubernetes is an open-source container-orchestration system for automating application deployment, scaling, and management across clusters of hosts. It works with a range of container tools, including Docker. Many cloud services offer a Kubernetes-based platform or infrastructure as a service

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(PaaS or IaaS) on which Kubernetes can be deployed as a platform-providing service.
Kubernetes defines a set of building blocks (primitives), which collectively provide mechanisms that deploy, maintain, and scale applications based on CPU, memory or custom metrics. Through the Kubernetes API, it is loosely coupled and extensible to meet different workloads. The platform exerts its control over compute and storage resources by defining resources as Objects, which can then be managed as such. The key objects are:

- Pod: It is a higher level of abstraction grouping containerised components. It consists of one or more containers that are guaranteed to be co-located on the host machine and can share resources. The basic scheduling unit in Kubernetes is a pod. Every pod has an IP. We can optionally specify CPU and memory (RAM) needed for each container. Pods are hosted inside a node, which can be a physical machine, virtual machine (VM), VM on cloud.
- Replica Sets: They are a grouping mechanism that lets Kubernetes maintain the number of instances that have been declared for a given pod.
- Services: A service is a set of pods that work together, such as one tier of a multi-tier application. The set of pods that constitute a service are defined by a label selector.
- Volumes: A volume provides persistent storage that exists for the lifetime of the pod itself. Kubernetes allows you to choose the storage system of your choice: local, cloud (AWS), network (NFS).
- ConfigMaps and Secrets: The former helps deciding where to store and manage configuration information, while the latter helps with sensitive data. It provides two closely related mechanisms to deal with this need: "configmaps" and "secrets", both of which allow for configuration changes to be made without requiring an application build. They are both objects created outside pods and containers. They are stored in ETCD (key-value datastore).
- StatefulSets: It helps scale Stateful workloads, because the state needs to be preserved if a pod is restarted, and if the application is scaled up or down, then the state may need to be redistributed. StatefulSets are controllers that enforce the properties of uniqueness and ordering amongst instances of a pod and can be used to run Stateful applications.
- DaemonSets: For some use cases, one need to run a pod on every single node in the cluster. This is useful for use cases like log collection, and storage services. DaemonSets do this kind of pod scheduling.

Kubernetes provides some mechanisms that allow one to manage, select, or manipulate its objects. For instance, it enables clients (users or internal components) to attach keys called labels to any API object in the system, such as pods and nodes. A Replica Set declares the number of instances of a pod that is needed, and a Replication Controller manages the system so that the number of healthy pods that are running matches the number of pods declared in the Replica Set (determined by evaluating its selector). The design principles underlying Kubernetes have been applied to developing a solution that allows one to create, configure, and manage Kubernetes clusters. This function is exposed via an API called the Cluster API.
We now present a high value overview of Kubernetes architecture. See Figure (1.20). A master node manages a set of worker nodes (hosting the pods), forming a team or cluster (at minimum one master node and one worker node). We can have multiple clusters. The master node is responsible for managing the cluster by monitoring nodes and pods in the cluster. When a node fails, it moves the workload of the failed node to another worker node. The master node has four components:

- API server: for all communications (Json over HTTP API). It exposes API for almost every operation.
- Scheduler: schedules pods across multiple nodes.
- Controller manager: runs controllers: node controller, replication controller, endpoints controller, service account and token controllers.
- ETCD: open source, distributed key-value database from CoreOS. It is access via the API server.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Every worker node in a Kubernetes cluster must run a container runtime like Docker. It has three components

- Kubelet: It is an agent running on each node which communicates with components from the master node. It makes sure that containers are running in a pod. In case of a problem with a pod, it will try to restart the pods on the same node or a different node.
- Kube-proxy: It is a network agent which runs on each node responsible for maintaining network configuration and rules. It exposes services to the outside world.
- Container runtime: It is a software responsible for running containers. Several platform choices exist: Docker, Containerd, Cri-o, Rktlet, Kubernetes CRI.

Figure 1.20: ML Kubernetes Architecture

1.7.3.1.2 Big data tools

- Apache Spark is an open-source distributed general-purpose cluster-computing framework. It is a fast and general engine for large-scale data processing. It is a distributed data-processing framework providing an interface for programming entire clusters with implicit data parallelism and fault tolerance. Its architectural foundation is the Resilient Distributed Dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. Spark facilitates the implementation of both iterative algorithms, which visit their data set multiple times in a loop, and interactive/exploratory data analysis, that is, the repeated database-style querying of data.
  Apache Spark requires 1) a cluster manager and 2) a distributed storage system. For cluster management, Spark supports standalone (native Spark cluster), Hadoop YARN, Apache Mesos or Kubernetes. For distributed storage, Spark can interface with a wide variety, including Alluxio, Hadoop Distributed File System (HDFS), MapR File System (MapR-FS), Cassandra, OpenStack Swift, Amazon S3, Kudu, Lustre file system, or a custom solution can be implemented. See Figure (1.21).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 1.21: ML Spark Diagram

Apache Spark provides the following key features:

- Include pre-built machine-learning algorithms and graph analysis algorithms that are especially written to execute in parallel and in memory.
- Run programs up to $100X$ faster than Hadoop MapReduce in memory, or $10X$ faster on disk.
- Write applications quickly in Java, Scala or Python.
- Combine SQL, streaming, and complex analytics.

Spark Streaming uses Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD transformations on those mini-batches of data. Structured Streaming APIs enable building end-to-end streaming applications called continuous applications in a consistent, fault-tolerant manner that can handle all of the complexities of writing such applications. It does so without having to reason about the details of streaming itself and by allowing the usage of familiar concepts within Spark SQL such as DataFrames and Datasets. It is also integrated with third party components such as Kafka, HDFS, S3, RDBMS, etc.
When using Kubernetes as a cluster manager, the submission mechanism works as follows:

- Spark creates a Spark driver running within a Kubernetes pod.
- The driver creates executors which are also running within Kubernetes pods and connects to them, and executes application code.
- When the application completes, the executor pods terminate and are cleaned up, but the driver pod persists logs and remains in completed state in the Kubernetes API until it is eventually garbage collected or manually cleaned up.

• Apache Kafka is a unified, high-throughput, low-latency platform for handling real-time data feeds. It is a messaging and integration platform acting as the central hub to stream data or process in batch from a source. This is a production ready open source solution which is horizontally scalable, fault-tolerant, and low latency. It can connect to external systems (for data import/export) via Kafka Connect and provides Kafka Streams, a Java stream processing library. Kafka runs on a cluster of one or more servers. For stream processing, Kafka offers the Streams API that allows writing Java applications that consume data from Kafka and write results back to Kafka. Apache Kafka also works with external stream processing systems such as Apache Apex, Apache Flink, Apache Spark, and Apache Storm. For example, once the data is processed, Spark Streaming can publish results into another Kafka topic, or store in HDFS, databases or dashboards. See Figure (1.22).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 1.22: Kafka Spark Diagram

- Attunity - Databricks: Databricks develops a web-based platform for working with Spark, that provides automated cluster management and IPython-style notebooks. Attunity is a unified platform integrating data and using delta lake open source to process big data. It makes the data reliable for analytics by streaming, updating/deleting, batching data. It uses Spark and, through notebooks, allows for fast debugging and data analysis. The automate streaming data pipelines is as follows:
  - Make data available through CDC and automated refinement. Use Spark transforms to generate continuous data.
  - create analytic ready data sets in databricks unified analytics (DUA) platform.

1.7.3.1.3 Pipeline creator: workflow manager

- Kubeflow is a free and open-sourced project designed to make running Machine Learning workflows on Kubernetes clusters simpler and more coordinated. It combines common open source projects into one big ecosystem by defines standards to create these pipelines. This is a cloud-native framework for employing Machine Learning in containerised environments in Kubernetes. It started as an internal Google project to run TensorFlow jobs on Kubernetes in a simpler and easier way. It was based specifically on the TensorFlow Extended (TFX) pipeline.
  At its core, Kubeflow offers an end-to-end orchestration toolkit to build on Kubernetes as a way to deploy, scale and manage complex systems. Features such as running JupyterHub servers allowing multiple users to contribute to a project simultaneously has become an invaluable asset of Kubeflow. Detailed management of a project and in depth monitoring/analysing of said project are paramount attributes in Kubeflow. It allows researchers to develop a complete pipeline composed of segmented steps, which are loosely coupled components of an ML pipeline. This way, pipelines become easily reusable and modifiable for other jobs. Thus, one can save an incalculable amount of labour necessary to develop a new data pipeline for each specific use case.

- Apache Airflow is an open-source workflow management platform (system) started at Airbnb (see Beauchemin [2015]). It is a platform to programmaticaly author, schedule and monitor data pipelines. It relies on four core elements that allow it to simplify any given pipeline:
  1. DAGs: It uses Directed Acyclic Graphs (see Section (13.1)) to manage workflow orchestration. It allows for a large number of possibilities to structure the pipeline.
  2. Tasks: DAGs are divided into tasks, and all of the work happens through the code written in these task. Tasks and dependencies are defined in Python and then Airflow manages the scheduling and execution.
  3. Runs: Airflow has its own scheduler. DAGs can be run either on a defined schedule or based on external event triggers.
  4. X-COM: It allows to pass information between the multiple tasks.

  The rich user interface (UI) makes it easy to visualise pipelines running in production, monitor progress and troubleshoot issues when needed.
  Some of the features offered by Airflow are:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Dynamic: Airflow pipelines are configuration as code (Python), allowing for dynamic pipeline generation. This allows for writing code that instantiate pipelines dynamically.
- Extensible: Easily define your own operators, executors and extend the library so that it fits the level of abstraction that suits your environment.
- Elegant: Airflow pipelines are lean and explicit. Parameterising your scripts is built in the core of Airflow using powerful Jinja templating engine.

1.7.3.1.4 Serverless platform Serverless computing is a cloud-computing execution model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. It can simplify the process of deploying code into production. Scaling, capacity planning and maintenance operations may be hidden from the developer or operator. Serverless code can be used in conjunction with code deployed in traditional styles, such as microservices. Alternatively, applications can be written to be purely serverless and use no provisioned servers at all. Most, but not all, serverless vendors offer compute runtimes, also known as function as a service (FaaS) platforms, which execute application logic but do not store data. AWS Lambda, introduced by Amazon in 2014, was the first public cloud infrastructure vendor with an abstract serverless computing offering. Google Cloud Platform offers Google Cloud Functions since 2016. Microsoft Azure offers Azure Functions. Oracle introduced Fn Project, an open source serverless computing framework offered on Oracle Cloud Platform and available on GitHub for deployment on other platforms. OpenWhisk is the core technology in IBM Cloud Functions.
Several serverless databases emerged, extending the serverless execution model to the RDBMS, eliminating the need to provision or scale virtualised or physical database hardware.

- Nuclio: It is an open source and managed serverless platform used to minimise development and maintenance overhead and automate the deployment of data-science based applications. It collect data from multiple sources, prepare and explore, accelerate and automate training, deploy in a click. Its functions are as follows:
  - Fast: Real-time performance running up to $400,000$ function invocations per second.
  - No lock-ins: Portable across low laptops, edge, on-prem and multi-cloud deployments.
  - GPU support: Support GPUs for optimized utilization and sharing.
  - Fast deployment: Automated deployment to production in a few clicks from Jupyter notebook.

1.7.3.1.5 Notebook A notebook interface is a virtual notebook environment used for literate programming. It pairs the functionality of word processing software with both the shell and kernel of that notebook's programming language. It allows to document research procedures, data, calculations, and findings. It tracks methodology to make it easier to reproduce results and calculations with different data sets.

- Project Jupyter (Julia, Python and R) is a non-profit organisation created to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.
  - The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. It supports execution environments in over 40 programming languages. It can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer. It allows to leverage big data tools, such as Apache Spark, from Python, R and Scala. The Jupyter Notebook has become a popular user interface for cloud computing.
    Uses include: data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more.
  - JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data. JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning. JupyterLab is extensible and modular: write plugins that add new components and integrate with existing ones.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Apache Zeppelin: It is a web-based notebook that enables data-driven, interactive data analytics and collaborative documents with SQL, Scala and more. It allows you to visualise your data dynamically and in real-time. Through the forms that you can create within a Zeppelin dashboard you can easily create dynamic scripts that use the forms' input to run a specific set of operations on a dynamically specified dataset. In addition, Zeppelin offers a huge number of interpreters allowing its notes to run multiple types of scripts.

1.7.3.2 The platform

Users deploying Hadoop in the cloud usually find it much simpler and cheaper to run it over cloud storage layers such as Amazon S3 or managed databases such as Amazon RedShift and Google BigQuery. However, YARN limits users to Hadoop and Java focused tools, while nowadays scientists are using microservices and Python-based tools. YARN was developed to run isolated Java processes to process big data workload and then improved to support Docker containers. If we choose to outsource IT operations to public cloud, one solution is to use Kubernetes as our orchestration layer. It can host different services including big data tools (Apache Spark or Presto), data-science and AI tools (Jupyter, TensorFlow, PyTorch, etc.) and any other application or data microservice. For instance, AI technologies led to the evolution of serverless technologies in which a developer submits code and requirements and the serverless platform automates the deployment, scaling and management of the application. Examples are OpenWhisk, Nuclio, and Fn designed to run over Kubernetes. For data, one can keep using managed cloud storage systems and databases, which are cheaper and simpler to maintain than the Hadoop file system. One advantage of Kubernetes is its portability, enabling users to build clusters which span multiple clouds or are distributed across locations. It facilitates the development or testing of microservices in the cloud and deployment in one or many edge locations automatically.

1.7.3.2.1 Data pipeline management We present a platform that can manage the data pipeline, the development of machine learning models, and the model deployment pipeline. At a high level, the infrastructure will consist of managed microservices:

1. Data
    - Define and stream any data source: external (public data) or within the internal IoT system.
2. Orchestration: Kubernetes
3. Middleware
    - Big data
        - Store all data in data buckets in the cloud.
        - Store some of the data in high cache databases.
    - Serverless technologies
        - Automating the deployment, scaling and management of the application.
    - AI analysis
        - Define and run data science workflows.
        - Train and deploy machine learning and artificial intelligence models. This is the ML-model. One can use TensorFlow or Neptune, among others.
        - Augment insight with existing workflows.

One way of building such an architecture is to use Kubernetes as a cluster manager, making the system cloud agnostic and on-premise deployable. We consider Spark's Structured Streaming APIs integrated with Kafka for consuming messages, performing complex windowing ETL (database function extract, transform, and load), and pushing the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

desired output to various sinks such as memory, console, file, databases, and back to Kafka.
Figure (1.23) gives an overview of the high level architecture for data piping and machine learning models (ML-model). It is a schematic diagram of the platform managing any arbitrary data source by creating real-time data streams with both Kafka and Spark and storing this data on the cloud. Any user will be able to connect securely to the infrastructure via Jupyter lab notebook (or Apache Zeppelin) and run data science procedure such as feature engineering and feature selections. The Jupyter lab users will also be able to define bespoke machine learning pipelines which will manage the training and deployment of models. A User Interface (such as DataDog) will augment the users role with actionable and decision processes which will be stored for future analysis.

Figure 1.23: ML Production Framework

1.7.3.2.2 Data infrastructure This system is able to define an arbitrary data source and stream that data in real-time. The data pipeline have to stream, store, analyse, and pre-process. This is shown in the first half of Figure (1.23), where Kafka creates data stream from the source, Spark runs real-time analysis on the data and store data to the cloud. Kafka is used for durable and scalable ingestion of streams of events coming from many producers to many consumers. Spark is used for processing large amounts of data, including real-time and near-real-time streams of events. They are combined to reach our goal as follows:

- A Kafka pipeline can be securely created for each IoT devices and internal/external data sources. Kafka makes requests to data source and the sources make a response (or non-response) which is streamed to an end-point. At the end-point, this is ingested into the cloud and a lambda split is formed where data is stored in to buckets/databases and streamed directly to a spark application (which manages the data infrastructure).
- Spark acts as an in-memory layer that sits above multiple data stores, where data can be loaded into memory and analysed in parallel across a cluster.

1.7.3.2.3 ML model infrastructure Once the data infrastructure is implemented, ML-model manages all microservices for model training and deployment. For instance, one can use TensorFlow or Neptune, among others. This is implemented on a Kubernetes platform to manage container orchestration and also make the system agnostic

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

to cloud provider, and even deployable to on-prem servers.
This infrastructure can either be delivered as PaaS microservice, or, we can install it on cloud provider, which will be managed with UI such as DataDog or a custom built UI.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 2

Information and knowledge

2.1 Algebraic logic

See books by Koppelberg [1989], Givant et al. [2009], among other.

2.1.1 Boolean algebra

Boolean algebra as the calculus of two values is fundamental to computer circuits, computer programming, and mathematical logic, and is also used in other areas of mathematics such as set theory and statistics.

2.1.1.1 An overview

Introduced by George Boole [1847/2016] and extended in Boole [1854/2003], Boolean algebra is the branch of algebra in which the values of the variables are the truth values true and false, usually denoted 1 and 0 respectively. Instead of elementary algebra where the values of the variables are numbers, and the prime operations are addition and multiplication, the main operations of Boolean algebra are the conjunction And denoted as $\wedge$, the disjunction Or denoted as $\vee$, and the negation Not denoted as $\neg$. Thus, it is a formalism for describing logical relations in the same way that elementary algebra describes numeric relations. Boolean algebra has been fundamental in the development of digital electronics, and is used in all modern programming languages as well as in set theory and statistics.
The work of Boole was extended and refined by a number of authors such as Jevons [1874]. Augustus De Morgan had worked on the logic of relations, and Charles Sanders Peirce integrated his work with Boole's during the 1870s (see Peirce [1931]). In 1937 Shannon wrote a master's thesis, at the Massachusetts Institute of Technology, in which he showed how Boolean algebra could optimise the design of systems of electromechanical relays then used in telephone routing switches. He also proved that circuits with relays could solve Boolean algebra problems. He applied the rules of Boole's algebra in switching circuits, and he introduced switching algebra as a way of analysing and designing circuits by algebraic means in terms of logic gates. A version of the paper was published in the 1938 issue of the Transactions of the American Institute of Electrical Engineers (see Shannon [1938]). Note, Victor Shestakov, at Moscow State University, proposed a theory of electric switches based on Boolean logic even earlier than Claude Shannon, which was published in 1941. Employing the properties of electrical switches to process logic is the basic concept that underlies all modern electronic digital computers. Nowadays, the utilisation of the binary properties of electrical switches to perform logic functions is the basic concept that underlies all electronic digital computer designs. Thus, in circuit engineering, switching algebra and Boolean algebra are often used interchangeably (see Balabanian et al. [2001]). Efficient implementation of Boolean functions is a fundamental problem in the design of combinational logic circuits. For instance, modern electronic design automation tools for VLSI circuits often rely on an efficient representation of Boolean functions known as (reduced ordered) binary decision diagrams (BDD) for logic synthesis and formal verification.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Since logic sentences that can be expressed in classical propositional calculus have an equivalent expression in Boolean algebra, Boolean logic is sometimes used to denote propositional calculus performed in this way (see Parkes [2002]). However, Boolean algebra is not sufficient to capture logic formulas using quantifiers, like those from first order logic. Although the development of mathematical logic did not follow Boole's program, the connection between his algebra and logic was later put on firm ground in the setting of algebraic logic, which also studies the algebraic systems of many other logics. The problem of determining whether the variables of a given Boolean (propositional) formula can be assigned in such a way as to make the formula evaluate to true is called the Boolean satisfiability problem (SAT), and is of importance to theoretical computer science, being the first problem shown to be NP-complete. The closely related model of computation known as a Boolean circuit relates time complexity (of an algorithm) to circuit complexity.

2.1.1.2 Description

Boolean algebra consider the truth values false and true which are represented with the bits (or binary digits), namely 0 and 1. They do not behave like the integers, for which $1 + 1 = 2$, but may be identified with the elements of the two-element field GF(2), that is, integer arithmetic modulo 2, for which $1 + 1 = 0$. Addition and multiplication then play the Boolean roles of XOR (exclusive-or) and AND (conjunction) respectively, with disjunction $x \vee y$ (inclusive-or) definable as $x + y - xy$. Boolean algebra also deals with functions which have their values in the set $\{0, 1\}$. A sequence of bits is a commonly used such function. Another common example is the subsets of a set E: to a subset F of E is associated the indicator function that takes the value 1 on F and 0 outside F. The most general example is the elements of a Boolean algebra, with all of the foregoing being instances thereof.

2.1.1.2.1 Some operations The basic operations of Boolean algebra are as follows:

- AND (conjunction), denoted $x \wedge y$ (sometimes x AND y or Kxy), satisfies $x \wedge y = 1$ if $x = y = 1$, and $x \wedge y = 1$ otherwise.
- OR (disjunction), denoted $x \vee y$ (sometimes x OR y or Axy), satisfies $x \vee y = 0$ if $x = y = 0$, and $x \vee y = 1$ otherwise.
- NOT (negation), denoted $\neg x$ (sometimes NOT x, Nx or $!x$), satisfies $\neg x = 0$ if $x = 1$, and $\neg x = 1$ if $x = 0$.

Alternatively the values of $x \wedge y$, $x \vee y$, and $\neg x$ can be expressed by tabulating their values with truth tables as in Figure (2.1).

x	y	$x \wedge y$	$x \vee y$
0	**0**	0	0
1	**0**	0	1
0	**1**	0	1
1	**1**	1	1

x	$\neg x$
0	1
1	0

Figure 2.1: Boolean operations

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

If the truth values 0 and 1 are interpreted as integers, these operations may be expressed with the ordinary operations of arithmetic (where $x + y$ uses addition and xy uses multiplication), or by the minimum/maximum functions:

$$\begin{aligned} x \wedge y &= xy = \min(x, y) \\ x \vee y &= x + y - xy = \max(x, y) \\ \neg x &= 1 - x \end{aligned}$$

One may consider that only the negation and one of the two other operations are basic, because of the following identities that allow to define the conjunction in terms of the negation and the disjunction, and vice versa:

$$\begin{aligned} x \wedge y &= \neg(\neg x \vee \neg y) \\ x \vee y &= \neg(\neg x \wedge \neg y) \end{aligned}$$

Secondary operations: The three Boolean operations above are called basic because they can be taken as a basis for other Boolean operations that can be built up from them by composition, the manner in which operations are combined or compounded. Operations composed from the basic operations include the following examples:

$$\begin{aligned} x \to y &= \neg x \vee y \\ x \oplus y &= (x \vee y) \wedge \neg(x \wedge y) = (x \wedge \neg y) \vee (\neg x \wedge y) \\ x \equiv y &= \neg(x \oplus y) = (x \wedge y) \vee (\neg x \wedge \neg y) \end{aligned}$$

These definitions give rise to the truth tables in Figure (2.2) giving the values of these operations for all four possible inputs.

x	y	$x \to y$	$x \oplus y$	$x \equiv y$
0	**0**	1	0	1
1	**0**	0	1	0
0	**1**	1	1	0
1	**1**	1	0	1

Figure 2.2: Boolean secondary operations

The first operation $x \to y$ or Cxy is called material implication. If x is true then the value of $x \to y$ is taken to be that of y. But if x is false then the value of y can be ignored. However, the operation must return some boolean value and there are only two choices. So by definition, $x \to y$ is true when x is false.
The second operation $x \oplus y$ or Jxy is called exclusive or (often abbreviated as XOR) to distinguish it from disjunction as the inclusive kind. It excludes the possibility of both x and y. Defined in terms of arithmetic it is addition modulo 2 where $1 + 1 = 0$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The third operation, the complement of exclusive or, is equivalence or Boolean equality: $x \equiv y$ or Exy is true just when x and y have the same value. Since $x \oplus y$ complements, it can be understood as $x \neq y$ which is true when x and y are different. Equivalence's counterpart in arithmetic modulo 2 is $x + y + 1$. For example, given two operands, each with two possible values, there are $2^2 = 4$ possible combinations of inputs. Since each output can have two possible values, there are a total of $2^4 = 16$ possible binary Boolean operations.

2.1.1.2.2 Some laws A law of Boolean algebra is an identity between two Boolean terms, where a Boolean term is defined as an expression built up from variables and the constants 0 and 1 using the operations $\wedge$, vee, and $\neg$. The concept can be extended to the secondary operators. Boolean algebra satisfies many of the same laws as ordinary algebra when one matches up $\vee$ with addition and $\wedge$ with multiplication. Some common laws for conjunction and disjunction between the two algebras are:

- Associativity of $\vee$: $x \vee (y \vee z) = (x \vee y) \vee z$.
- Associativity of $\wedge$: $x \wedge (y \wedge z) = (x \wedge y) \wedge z$.
- Commutativity of $\vee$: $x \vee y = y \vee x$.
- Commutativity of $\wedge$: $x \wedge y = y \wedge x$.
- Distributivity of $\wedge$ over $\vee$: $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$.
- Identity of $\vee$: $x \vee 0 = x$.
- Identity of $\wedge$: $x \wedge 1 = x$.
- Annihilator of $\wedge$: $x \wedge 0 = 0$.

Some laws for conjunction and disjunction which hold in Boolean algebra but not in ordinary algebra are:

- Annihilator of $\vee$: $x \vee 1 = 1$.
- Idempotence of $\vee$: $x \vee x = x$.
- Idempotence of $\wedge$: $x \wedge x = x$.
- Absorption 1: $x \wedge (x \vee y) = x$.
- Absorption 2: $x \vee (x \wedge y) = x$.
- Distributivity of $\vee$ over $\wedge$: $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$.

Note, these laws for conjunction and disjunction are monotone since changing either argument either leaves the output unchanged or the output changes in the same way as the input, and changing any variable from 0 to 1 never results in the output changing from 1 to 0.
In the case of non-monotone laws we get the complement operation:

- Complementation 1: $x \wedge \neg x = 0$.
- Complementation 2: $x \vee \neg x = 1$.

These complementation laws together with the monotone laws suffice to define Boolean algebra and can therefore be taken as one possible complete set of laws or axiomatisation of Boolean algebra. Every law of Boolean algebra follows logically from these axioms. Furthermore, Boolean algebras can then be defined as the models of these axioms. Note, this axiomatisation is not the only one.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2.1.1.2.3 Representations Digital logic is the application of the Boolean algebra of 0 and 1 to electronic hardware consisting of logic gates connected to form a circuit diagram (see Shannon [1949b]). Each gate implements a Boolean operation, and is depicted schematically by a shape indicating the operation. For instance, the shapes associated with the gates for conjunction (AND-gates), disjunction (OR-gates), and complement (inverters) are displayed in Figure (2.3).

Figure 2.3: Boolean logic gates

The lines on the left of each gate represent input wires or ports. The value of the input is represented by a voltage on the lead. For the active-high logic, 0 is represented by a voltage close to zero or ground, while 1 is represented by a voltage close to the supply voltage; active-low reverses this. The line on the right of each gate represents the output port, which normally follows the same voltage conventions as the input ports. Complement is implemented with an inverter gate. The triangle denotes the operation that simply copies the input to the output; the small circle on the output denotes the actual inversion complementing the input. The convention of putting such a circle on any port means that the signal passing through this port is complemented on the way through, whether it is an input or output port. The Duality Principle, or De Morgan's laws, can be understood as asserting that complementing all three ports of an AND gate converts it to an OR gate and vice versa. See Figure (2.4).

Figure 2.4: Boolean De Morgan equivalents

2.1.1.2.4 About algebras A concrete Boolean algebra or field of sets is any non-empty set of subsets of a given set X closed under the set operations of union, intersection, and complement relative to X. A subset Y of X can be identified with an indexed family of bits with index set X, with the bit indexed by $x \in X$ being 1 or 0 according to whether or not $X \in Y$. For example, if $X = \{a, b, c\}$ where a, b, c are viewed as bit positions in that order from left to right, the eight subsets $\{\}$, $\{c\}$, $\{b\}$, $\{b, c\}$, $\{a\}$, $\{a, c\}$, $\{a, b\}$, and $\{a, b, c\}$ of X can be identified with the respective bit vectors 000, 001, 010, 011, 100, 101, 110, and 111. Bit vectors indexed by the set of natural numbers are infinite sequences of bits, while those indexed by the reals in the unit interval $[0, 1]$ are packed too densely to be able to write conventionally but nonetheless form well-defined indexed families (imagine colouring every point of the interval $[0, 1]$ either black or white independently. The black points then form an arbitrary subset of $[0, 1]$). Thus, from this bit vector viewpoint, a concrete Boolean algebra can be defined equivalently as a non-empty set of bit vectors all of the same length (more generally, indexed by the same set) and closed under the bit vector operations of bitwise $\vee$, $\wedge$, and $\neg$. The set $\{0, 1\}$ and its Boolean operations as treated above can be understood as the special case of bit vectors of length one, which by the identification of bit vectors with subsets can also be understood as the two subsets of a one-element

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

set. This is called the prototypical Boolean algebra.
Rather than showing that the Boolean laws are satisfied, we can instead postulate a set X, two binary operations on X, and one unary operation, and require that those operations satisfy the laws of Boolean algebra. The elements of X need not be bit vectors or subsets but can be anything at all, which leads to the more general abstract definition: A Boolean algebra is any set with binary operations $\vee$ and $\wedge$, and a unary operation neg satisfying the Boolean laws. This axiomatic definition of a Boolean algebra as a set and certain operations satisfying certain laws or axioms by flat is entirely analogous to the abstract definitions of group, ring, field characteristic of modern or abstract algebra. Note, the following is therefore an equivalent definition: A Boolean algebra is a complemented distributive lattice. Further, we consider all Boolean laws which can be defined as all equations that hold for the Boolean algebra of 0 and 1. However, here are infinitely many such laws. One solution is to require only finitely many laws such as the equations defined above. In that case, Boolean algebra is finitely axiomatisable or finitely based. Simplifying further, as done just above, we get the traditional axiomatisation of Boolean algebra as a complemented distributive lattice.

2.1.1.2.5 Propositional logic Propositional logic is a logical system that is intimately connected to Boolean algebra (see Givant et al. [2009]). Many syntactic concepts of Boolean algebra carry over to propositional logic with only minor changes in notation and terminology, while the semantics of propositional logic are defined via Boolean algebras in a way that the tautologies (theorems) of propositional logic correspond to equational theorems of Boolean algebra.
Every Boolean term corresponds to a propositional formula of propositional logic. For instance, Boolean variables $x, y, ...$ become propositional variables (or atoms) $P, Q, ...$ Further, Boolean terms such as $x \vee y$ become propositional formulas $P \vee Q$, 0 becomes False or $\perp$, and 1 becomes True or T. In general, generic propositions are denoted with Greek letters $\Phi, \Psi, ...$ The semantics of propositional logic rely on truth assignments: The propositional variables are mapped to elements of a fixed Boolean algebra, and then the truth value of a propositional formula using these letters is the element of the Boolean algebra that is obtained by computing the value of the Boolean term corresponding to the formula. In classical semantics, only the two-element Boolean algebra is used, while in Boolean-valued semantics arbitrary Boolean algebras are considered. A tautology is a propositional formula that is assigned truth value 1 by every truth assignment of its propositional variables to an arbitrary Boolean algebra (or, equivalently, every truth assignment to the two element Boolean algebra). These semantics permit a translation between tautologies of propositional logic and equational theorems of Boolean algebra. For instance, every tautology Φ of propositional logic can be expressed as the Boolean equation $\Phi = 1$, which will be a theorem of Boolean algebra. Conversely every theorem $\Phi = \Psi$ of Boolean algebra corresponds to the tautologies

$$\big(\Phi \vee \neg\Psi\big) \wedge \big(\neg\Phi \vee \Psi\big) \text{ and } \big(\Phi \wedge \Psi\big) \vee \big(\neg\Phi \wedge \neg\Psi\big)$$

If $\rightarrow$ is in the language, these last tautologies can also be written as

$$\big(\phi \rightarrow \Psi\big) \wedge \big(\Psi \rightarrow \phi\big)$$

or as two separate theorems

$$\phi \rightarrow \Psi \text{ and } \Psi \rightarrow \phi$$

If $\equiv$ is available then the single tautology $\phi \equiv \Psi$ can be used.
One application of propositional calculus is the analysis of propositions and deductive arguments in natural language. Whereas the proposition "if $x = 2$ then $x + 1 = 3$" depends on the meanings of such symbols as $+$ and 1, the proposition "if $x = 2$ then $x = 2$" does not; it is true merely by virtue of its structure, and remains true whether "$x = 2$" is replaced by "$x = 3$" or "the moon is made of blue cake." The generic or abstract form of this tautology is "if P then P", or in the language of Boolean algebra "$P \rightarrow P$". Replacing P by $x = 2$ or any other proposition is called instantiation of P by that proposition, and the result of instantiating P in an abstract proposition is called an instance of the proposition. Thus, "$X = 2 \rightarrow x = 2$" is a tautology by virtue of being an instance of the abstract tautology "$P \rightarrow P$". All occurrences of the instantiated variable must be instantiated with the same proposition,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

to avoid such nonsense as $P \to x = 2$ or $x = 2 \to x = 3$. Propositional calculus restricts attention to abstract propositions, those built up from propositional variables using Boolean operations. Instantiation is still possible within propositional calculus, but only by instantiating propositional variables by abstract propositions, such as instantiating Q by $Q \to P$ in $P \to \big(Q \to P\big)$ to yield the instance $P \to \big[\big(Q \to P\big) \to P\big]$.
An axiomatisation of propositional calculus is a set of tautologies called axioms and one or more inference rules for producing new tautologies from old. A proof in an axiom system A is a finite nonempty sequence of propositions each of which is either an instance of an axiom of A or follows by some rule of A from propositions appearing earlier in the proof (thereby disallowing circular reasoning).
Propositional calculus is commonly organized as a Hilbert system, whose operations are just those of Boolean algebra and whose theorems are Boolean tautologies, those Boolean terms equal to the Boolean constant 1. Another form is sequent calculus, which has two sorts, propositions as in ordinary propositional calculus, and pairs of lists of propositions called sequents, such as $A \vee B, A \wedge C, ..., \vdash A, B \to C, ...$

2.1.2 Fuzzy logic

2.1.2.1 An overview

Fuzzy logic is a form of many-valued logic in which the truth values of variables may be any real number between 0 and 1 inclusive. It is employed to handle the concept of partial truth, where the truth value may range between completely true and completely false. By contrast, in Boolean logic (see Section (2.1.1)), the truth values of variables may only be the integer values 0 or 1.
Fuzzy logic has been studied since the 1920s, as infinite-valued logic by Tarski [1942], among other. Later, Zadeh [1965] published his seminal work on fuzzy sets in which he detailed the mathematics of fuzzy set theory. Zadeh [1973] proposed the theory of fuzzy logic where he used a membership function (with a range covering the interval $[0, 1]$) operating on the domain of all possible values. He proposed new operations for the calculus of logic and showed that fuzzy logic was a generalisation of classical and Boolean logic. It is based on the observation that people make decisions based on imprecise and non-numerical information, fuzzy models or sets are mathematical means of representing vagueness and imprecise information, hence the term fuzzy. He also proposed fuzzy numbers as a special case of fuzzy sets, as well as the corresponding rules for consistent mathematical operations (fuzzy arithmetic). Various authors extended the concept of rule based expert systems to embrace fuzzy logic (see Tsukamoto [1979]).
While both degrees of truth and probabilities range between 0 and 1 and hence may seem similar at first, fuzzy logic uses degrees of truth as a mathematical model of vagueness, while probability is a mathematical model of ignorance. A basic application might characterise various sub-ranges of a continuous variable. However, in fuzzy logic applications, non-numeric values are often used to facilitate the expression of rules and facts. For instance, a linguistic variable such as age may accept values such as young and its antonym old. Since natural languages do not always contain enough value terms to express a fuzzy value scale, it is common practice to modify linguistic values with adjectives or adverbs. For example, we can use the hedges rather and somewhat to construct the additional values rather old or somewhat young.

2.1.2.2 Description

2.1.2.2.1 The process Fuzzification operations can map mathematical input values into fuzzy membership functions. And the opposite de-fuzzifying operations can be used to map a fuzzy output membership functions into a crisp output value that can be then used for decision or control purposes. The process is as follows

1. Fuzzify all input values into fuzzy membership functions.
2. Execute all applicable rules in the rulebase to compute the fuzzy output functions.
3. De-fuzzify the fuzzy output functions to get crisp output values.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2.1.2.2.2 The operators Fuzzy logic works with membership values in a similar way to Boolean logic. To this end, replacements for basic operators And, Or, Not must be available. There are several ways to this. A common replacement is called the Zadeh operators:

- And is defined as the minimum operation on two variables.
- Or is defined as the maximum operation on two variables.
- Not is the inversion of a single variable.

See Figure (2.5).

Boolean	Fuzzy
AND(x,y)	MIN(x,y)
OR(x,y)	MAX(x,y)
NOT(x)	1 – x

Figure 2.5: Zadeh operators

For $True/1$ and $False/0$, the fuzzy expressions produce the same result as the Boolean expressions. There are also other operators, more linguistic in nature, called hedges that can be applied. These are generally adverbs such as very, or somewhat, which modify the meaning of a set using a mathematical formula. However, an arbitrary choice table does not always define a fuzzy logic function.
Another set of AND/OR operators is based on multiplication

$$\begin{aligned} x \text{ And } y &= x * y \\ x \text{ Or } y &= 1 - (1-x) * (1-y) = x + y - x * y \end{aligned}$$

where the term $1 - (1-x) * (1-y)$ comes from

$$\begin{aligned} x \text{ Or } y &= Not\big(\text{ And } \big(Not(x), Not(y)\big)\big) \\ x \text{ Or } y &= Not\big(\text{ And } \big(1-x, 1-y\big)\big) \\ x \text{ Or } y &= Not\big((1-x) * (1-y)\big) \\ x \text{ Or } y &= 1 - (1-x) * (1-y) \end{aligned}$$

The If-Then rules map input or computed truth values to desired output truth values. For example:

- IF temperature IS very cold Then $fan - speed$ is stopped.
- IF temperature IS cold Then $fan - speed$ is slow.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- IF temperature IS warm Then $fan - speed$ is moderate.
- IF temperature IS hot Then $fan - speed$ is high.

Given a certain temperature, the fuzzy variable hot has a certain truth value, which is copied to the high variable. Should an output variable occur in several Then parts, then the values from the respective IF parts are combined using the Or operator.

2.1.2.2.3 Fuzzy sets Fuzzy sets are often defined as triangle or trapezoid-shaped curves, as each value will have a slope where the value is increasing, a peak where the value is equal to 1 (which can have a length of 0 or greater) and a slope where the value is decreasing. They can also be defined using a sigmoid function.
The fuzzy sets are usually used in two ways:

1. As absolute sets where the value of a variable is used as input. It has linguistic values such as high, low, medium, and so on.
2. As comparative sets where the difference between two variables is input to the set. It has linguistic values such close to, greater than, less than, and so on.

One common case of (1) is the standard logistic function defined as

$$f(x) = \frac{1}{1 + e^{-x}}$$

We can also consider the threshold function which is used to generate fuzzy sets with linguistic values such as greater than, less than, much larger than, large, small and so on. For $\gamma > \beta > \alpha$, where α and γ control the position and steepness of the threshold, respectively, we have

$$f(x) = \begin{cases} 0 \text{ if } x < \alpha \\ 2\left[\frac{x-\alpha}{\gamma-\alpha}\right]^2 \text{ if } \alpha \leqslant x < \beta \\ 1 - 2\left[\frac{x-\gamma}{\gamma-\alpha}\right]^2 \text{ if } \beta \leqslant x < \gamma \\ 1 \text{ if } x \geqslant \gamma \end{cases}$$

In the case of comparative sets (2), we let the parameter a and c control the width and position of the fuzzy set, respectively. For the pair (x_p, y_p), we have

$$y_p = \frac{1}{1 + \left(\frac{x_p - c}{a}\right)^4}$$

2.1.2.2.4 Defuzzification In the case of defuzzification we want to get a continuous variable from fuzzy truth values. It would be easy if the output truth values were exactly those obtained from fuzzification of a given number. However, since all output truth values are computed independently, in most cases they do not represent such a set of numbers. Thus, we need to decide for a number that matches best the intention encoded in the truth value. For example, for several truth values of $fan - speed$, an actual speed must be found that best fits the computed truth values of the variables slow, medium and so on. Even though there is no single algorithm for this purpose, a common algorithm is

1. For each truth value, cut the membership function at this value
2. Combine the resulting curves using the Or operator
3. Find the center-of-weight of the area under the curve

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4. The x position of this center is then the final output.

Since the fuzzy system output is a consensus of all of the inputs and all of the rules, fuzzy logic systems can be well behaved when input values are not available or are not trustworthy. Weightings can be optionally added to each rule in the rulebase and weightings can be used to regulate the degree to which a rule affects the output values. These rule weightings can be based upon the priority, reliability or consistency of each rule. These rule weightings may be static or can be changed dynamically, even based upon the output from other rules.

In mathematical logic, there are several formal systems of "fuzzy logic", most of which are in the family of t-norm fuzzy logics.

2.1.2.2.5 Difference with probability Fuzzy logic and probability address different forms of uncertainty. While both fuzzy logic and probability theory can represent degrees of certain kinds of subjective belief, fuzzy set theory uses the concept of fuzzy set membership, that is, how much an observation is within a vaguely defined set, and probability theory uses the concept of subjective probability, that is, likelihood of some event or condition. The concept of fuzzy sets was developed in the mid-twentieth century at Berkeley as a response to the lacking of probability theory for jointly modelling uncertainty and vagueness (see Mares [2006]). Zadeh argues that fuzzy logic is different in character from probability, and is not a replacement for it. He fuzzified probability to fuzzy probability and also generalised it to possibility theory. More generally, fuzzy logic is one of many different extensions to classical logic intended to deal with issues of uncertainty outside of the scope of classical logic, the inapplicability of probability theory in many domains, and the paradoxes of Dempster-Shafer theory.

2.2 Introduction to information theory

See textbooks by Shannon et al. [1949], MacKay [2003], among others.

2.2.1 Presenting a few concepts

2.2.1.1 Uncertainty

According to Shannon, information is any entity or form that provides the answer to a question of some kind, or resolves uncertainty. It is thus related to data and knowledge, as data represents values attributed to parameters, and knowledge signifies understanding of real things or abstract concepts. In the case of data, the existence of information is not necessarily coupled to an observer (for example, it exists beyond an event horizon), while in the case of knowledge, the information requires a cognitive observer.
Information is conveyed either as the content of a message or through direct or indirect observation. That which is perceived can be construed as a message in its own right, and in that sense, information is always conveyed as the content of a message. Information can be encoded into various forms for transmission and interpretation (for example, information may be encoded into a sequence of signs, or transmitted via a signal). It can also be encrypted for safe storage and communication.
Shannon proposed that information reduces uncertainty and therefore reduces entropy. A typical example of this principle is flipping a two-sided coin. When tossing a coin, we are uncertain about the two possible outcomes. By observing the outcome (say, heads), we gain information that reduces the uncertainty about this event. The uncertainty of an event is measured by its probability of occurrence and is inversely proportional to that. The more uncertain an event, the more information is required to resolve uncertainty of that event. Shannon built on the work of fellow researchers Ralph Hartley and Harry Nyquist to reveal that coding and symbols were the key to resolving whether two sides of a communication had a common understanding of the uncertainty being resolved. Following the example of the coin he concluded that any type of information, could be encoded as a series of fundamental yes or no answers (see

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Section (2.1.1)). Nowadays this is known as bits of digital information, ones and zeroes, that represent everything from email text, digital photos, compact disc music or high definition video. The concept of information became closely related to notions of constraint, communication, control, data, form, education, knowledge, meaning, understanding, mental stimuli, pattern, perception, representation, and entropy.

2.2.1.2 Coding

Information theory (IT) studies the transmission, processing, extraction, and utilisation of information. Abstractly, information can be thought of as the resolution of uncertainty. In the case of communication of information over a noisy channel, this abstract concept was made concrete by Shannon [1948], in which "information" is thought of as a set of possible messages, where the goal is to send these messages over a noisy channel, and then to have the receiver reconstruct the message with low probability of error, in spite of the channel noise. Shannon introduced information theory (IT) by proposing some fundamental limits to the representation and transmission of information. In the noisy-channel coding theorem, he showed that, in the limit of many channel uses, the rate of information that is asymptotically achievable is equal to the channel capacity, a quantity dependent merely on the statistics of the channel over which the messages are sent.
Since then, information theory (IT) is a broad and deep mathematical theory, with equally broad and deep applications, amongst which is the vital field of coding theory. It has provided the theoretical motivations for many of the outstanding advances in digital communications and digital storage. Coding theory is concerned with finding explicit methods, called codes, for increasing the efficiency and reducing the error rate of data communication over noisy channels to near the channel capacity. These codes can be roughly subdivided into data compression (source coding) and error-correction (channel coding) techniques.
One generally uses the following digital communication model in the transfer of information:

1. The source is the source of (digital) data
2. The source encoder serves the purpose of removing as much redundancy as possible from the data. It is called the data compression portion.
3. The channel coder puts a modest amount of redundancy back in order to perform error detection or correction.
4. The channel is what the data passes through, possibly becoming corrupted along the way.
5. The channel decoder performs error correction or detection.
6. The source decoder undoes what is necessary to recover the data back.

There are other blocks that could be inserted in that model, such as

- a block enforcing channel constraints sometime called a line coder.
- a block performing encryption/decryption.
- a block performing lossy compression.

2.2.1.3 Randomness

One of the key concept in IT is that information is conveyed by randomness, that is, information is defined in some mathematical sense, which is not identical to the one used by humans. However, it is not too difficult to make the connection between randomness and information. Another important concept in IT is that of typical sequences. In a sequence of bits of length n, there are some sequences which are typical. For example, in a sequence of coin-tossing outcomes for a fair coin, we would expect the number of heads and tails to be approximately equal. A sequence not following this trend is thus atypical. A good part of IT is capturing this concept of typicality as precisely as possible

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

and using it to concluding how many bits are needed to represent sequence of data. The main idea being to use bits to represent only the typical sequences, since the other ones are rare events. This concept of typical sequences corresponds to the asymptotic equipartition property.
Shannon, for the first time, introduced the qualitative and quantitative model of communication as a statistical process underlying information theory. That is, information theory is based on probability theory and statistics. It concerns itself with measures of information of the distributions associated with random variables. Important quantities of information are entropy, a measure of information in a single random variable, and mutual information, a measure of information in common between two random variables. Given a discrete random variable X, with x a particular outcome occurring with probability (mass function) $p(x)$. Then, we assign to that event, x, the information that it conveys via the uncertainty measure

$$\text{uncertainty} \ = -\log p(x) \tag{2.2.1}$$

where the base of the logarithms determines the units of information. The units of $\log_2$ are in bits, that of $\log_e$ are in nats. For example, a random variable having two outcomes, 0 and 1, occurring with probability $p(0) = p(1) = \frac{1}{2}$, then each outcome conveys 1 bit of information. However, if $p(0) = 1$ and $p(1) = 0$, then the information conveyed when 0 happens is 0. We get no information from it, as we knew all along that it would happen. Hence, the information that 1 happens is ∞, as we are totally surprised by its occurrence. In general, one commonly use the average uncertainty provided by a random variable X taking value in a space χ, leading to the notion of entropy.
Much of the mathematics behind information theory with events of different probabilities were developed for the field of thermodynamics by Ludwig Boltzmann and J. Willard Gibbs. Connections between information-theoretic entropy and thermodynamic entropy, including the important contributions by Landauer in the 1960s (see Landauer [1961]), are explored in Entropy in thermodynamics and information theory.

2.2.2 Some facts on entropy in information theory

The entropy functional, defined on a Markov diffusion process, plays an important roll in the theory of information and statistical physics, informational macrodynamics and control systems. In information theory (IT), entropy is the average amount of information contained in each message received. That is, entropy is a measure of unpredictability of information content (see Equation (2.2.1)). Named after Boltzmann's H-theorem, Shannon [1948] defined the entropy H of a discrete random variable X with possible values $\{x_1, .., x_n\}$ and probability mass function $P(X)$ as

$$H(X) = E[I(X)] = E[-\log P(X)] \tag{2.2.2}$$

where $I(X)$ is the information content of X, which is itself a random variable. When taken from a finite sample, the entropy can explicitly be written as

Definition 2.2.1 *The entropy $H(X)$ of a discrete random variable X taking values $\{x_1, .., x_n\}$ is*

$$H(X) = \sum_{i=1}^{n} p(x_i)I(x_i) = -\sum_{i=1}^{n} p(x_i)\log_b p(x_i)$$

where b is the base of the logarithm used.

The unit of entropy is bit for $b = 2$, nat for $b = e$ (where e is Euler's number), and dit (or digit) for $b = 10$. Given the well-known limit $\lim_{p\to 0_+} p\log p = 0$, in the case where $P(x_i) = 0$ for some i, the value of the corresponding summand is taken to be 0. For example, taking a fair coin, we get

$$H(X) = -(\frac{1}{2}\log\frac{1}{2} + \frac{1}{2}\log\frac{1}{2}) = 1 \text{ bit}$$

For a biased coin with $p(0) = 0.9$, we get

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$H(X) = -(0.9\log 0.9 + 0.1\log 0.1) = 0.469 \text{ bit}$$

If X is a binary random variable

$$X = \begin{cases} 1 \text{ with probability } p \\ 0 \text{ with probability } 1-p \end{cases}$$

then the entropy of X is

$$H(X) = -p\log p - (1-p)\log(1-p)$$

For some function $g(X)$ of a random variable, we get $E[g(X)] = \sum_{x\in\mathcal{X}} g(x)p(x)$ so that for $g(x) = \log\frac{1}{p(x)}$ we get

$$H(X) = E[g(X)] = E[\log\frac{1}{p(X)}]$$

We are now interested in the entropy of pairs of random variables (X, Y).

Definition 2.2.2 *If X and Y are jointly distributed according to $p(X, Y)$, then the joint entropy $H(X, Y)$ is*

$$H(X,Y) = -E[\log p(X,Y)] = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x,y)$$

Definition 2.2.3 *If $(X, Y) \sim p(x, y)$, the conditional entropy of two events X and Y is given by*

$$H(Y|X) = -E_{p(x,y)}[\log p(Y|X)] = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y|x)$$

This quantity should be understood as the amount of randomness in the random variable Y given that you know the value of X. The conditional entropy can also be written as

$$\begin{aligned} E[Y|X] &= -\sum_{x\in\mathcal{X}} p(x)\sum_{y\in\mathcal{Y}} p(y|x)\log p(y|x) \\ &= -\sum_{x\in\mathcal{X}} p(x)H(Y|X=x) \end{aligned}$$

Theorem 2.2.1 *chain rule*

$$H(X,Y) = H(X) + H(Y|X)$$

We can also have a joint entropy with a conditioning on it.

***Corollary* 1**

$$H(X,Y|Z) = H(X|Z) + H(Y|X,Z)$$

The inspiration for adopting the word entropy in information theory came from the close resemblance between Shannon's formula and very similar known formulae from statistical mechanics. In statistical thermodynamics the most general formula for the thermodynamic entropy S of a thermodynamic system is the Gibbs entropy

$$S = -k_B\sum_i p_i \ln p_i$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where k_B is the Boltzmann constant, and p_i is the probability of a microstate. The Gibbs entropy translates over almost unchanged into the world of quantum physics to give the von Neumann entropy, introduced by John von Neumann in 1927

$$S = -k_B Tr(\rho \ln \rho)$$

where ρ is the density matrix of the quantum mechanical system and $Tr()$ is the trace. Note, at a multidisciplinary level, connections can be made between thermodynamic and informational entropy. In the view of Jaynes (1957), thermodynamic entropy, as explained by statistical mechanics, should be seen as an application of Shannon's information theory: the thermodynamic entropy is interpreted as being proportional to the amount of further Shannon information needed to define the detailed microscopic state of the system, that remains uncommunicated by a description solely in terms of the macroscopic variables of classical thermodynamics, with the constant of proportionality being just the Boltzmann constant.

2.2.3 Relative entropy and mutual information

Another useful measure of entropy that works equally well in the discrete and the continuous case is the relative entropy of a distribution. Given a random variable with true distribution p, which we do not know due to incomplete information, instead we assume the distribution q. Then, the code will need more bits to represent the random variable, and the difference in bits is denoted by $D(p||q)$. The relative entropy is defined as the Kullback-Leibler divergence from the distribution to the reference measure q as follows.

Definition 2.2.4 *The relative entropy or Kullback-Leibler distance between two probability mass functions $p(x)$ and $q(x)$ is defined as*

$$D_{KL}(p||q) = E_p[\log \frac{p(x)}{q(x)}] = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}$$

This form is not symmetric, and q appears only in the denominator. Alternatively, assume that a probability distribution p is absolutely continuous with respect to a measure q, that is, it is of the form $p(dx) = f(x)q(dx)$ for some non-negative q-integrable function f with q-integral 1, then the relative entropy can be defined as

$$D_{KL}(p||q) = \int \log f(x) p(dx) = \int f(x) \log f(x) q(dx)$$

In this form the relative entropy generalises (up to change in sign) both the discrete entropy, where the measure q is the counting measure, and the differential entropy, where the measure q is the Lebesgue measure. If the measure q is itself a probability distribution, the relative entropy is non-negative, and zero if $p = q$ as measures. It is defined for any measure space, hence coordinate independent and invariant under co-ordinate reparametrisations if one properly takes into account the transformation of the measure q. The relative entropy, and implicitly entropy and differential entropy, do depend on the reference measure q.

Another important concept, called mutual information, describes the amount of information a random variable tells about another one. That is, observing the output of a channel, we want to know what information was sent. The channel coding theorem is a statement about mutual information.

Definition 2.2.5 *Let X and Y be random variables with joint distribution $p(X,Y)$ and marginal distributions $p(x)$ and $p(y)$. The mutual information $I(X;Y)$ is the relative entropy between the joint distribution and the product distribution*

$$\begin{aligned} I(X;Y) &= D(p(x,y)||p(x)p(y)) \\ &= \sum_{x \in \mathcal{X}} p(x) \sum_{y \in \mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, when X and Y are independent, $p(x,y) = p(x)p(y)$, and we get $I(X;Y) = 0$. That is, in case of independence, Y, can not tell us anything about X. An important interpretation of mutual information comes from the following theorem.

Theorem 2.2.2

$$I(X;Y) = H(X) - H(X|Y)$$

which states that the information that Y tells us about X is the reduction in uncertainty about X due to the knowledge of Y. By symmetry we get

$$I(X;Y) = H(Y) - H(Y|X) = I(Y;X)$$

Using $H(X,Y) = H(X) + H(Y|X)$, we get

$$I(X;Y) = H(X) + H(Y) - H(X,Y)$$

The information that X tells about Y is the uncertainty in X plus the uncertainty about Y minus the uncertainty in both X and Y. We can summarise statements about entropy as follows

$$\begin{aligned}
I(X;Y) &= H(X) - H(X|Y) \\
I(X;Y) &= H(Y) - H(Y|X) \\
I(X;Y) &= H(X) + H(Y) - H(X,Y) \\
I(X;Y) &= I(Y;X) \\
I(X;X) &= H(X)
\end{aligned}$$

When dealing with the sequence of random variables $X_1, .., X_n$ drawn from the joint distribution $p(x_1, .., x_n)$, a variety of chain rules have been developed.

Theorem 2.2.3 *The joint entropy of $X_1, .., X_n$ is*

$$H(X_1, .., X_n) = \sum_{i=1}^{n} H(X_i|X_{i-1}, .., X_1)$$

The chain rule for entropy leads us to a chain rule for mutual information.

Theorem 2.2.4

$$I(X_1, .., X_n; Y) = \sum_{i=1}^{n} I(X_i; Y|X_{i-1}, ..., X_1)$$

2.2.4 Bounding performance measures

A large part of information theory consists in finding bounds on certain performance measures. One of the most important inequality used in IT is the Jensen's inequality. We are interested in convex functions because it is known that over the interval of convexity, there is only one minimum (for details see Appendix (23.2.2)). The following theorem describe the information inequality.

Theorem 2.2.5 $\log x \leqslant x - 1$, *with equality if and only if* $x = 1$.

We can now characterise some of the information measures defined above.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 2.2.6 *$D(p||q) \geqslant 0$, with equality if and only if $p(x) = q(x)$ for all x.*

Corollary 2 *Mutual information is positive, $I(X;Y) \geqslant 0$, with equality if and only if X and Y are independent.*

We let the random variable X takes values in the set $\mathcal{X}$, and we let $|\mathcal{X}|$ denotes the number of elements in that set. For discrete random variables, the uniform distribution over the range $\mathcal{X}$ has the maximum entropy.

Theorem 2.2.7 *$H(X) \leqslant \log|\mathcal{X}|$, with equality if and only if X has a uniform distribution.*

Note, if you can show that some performance criterion is upper-bounded by some function, then by showing how to achieve that upper bound, we have found an optimum. We see that the more we know, the less uncertainty there is.

Theorem 2.2.8 *Condition reduces entropy:*

$$H(X|Y) \leqslant H(X)$$

with equality if and only if X and Y are independent.

Theorem 2.2.9

$$H(X_1, ..., X_n) \leqslant \sum_{i=1}^{n} H(X_i)$$

with equality if and only if X_i are independent.

The following theorem allow us to deduce the concavity (or convexity) of many useful functions.

Theorem 2.2.10 *Log-sum inequality*
For non-negative numbers $a_1, ..., a_n$ and $b_1, .., b_n$,

$$\sum_{i=1}^{n} a_i \log \frac{a_i}{b_i} \geqslant (\sum_{i=1}^{n} a_i) \log \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} b_i}$$

with equality if and only if $\frac{a_i}{b_i} =$ constant .

The conditions on this theorem are much weaker than the one for Jensen's inequality, since it is not necessary to have the sets of numbers add up to 1. Using this inequality, one can prove a convexity statement about the relative entropy function.

Theorem 2.2.11 *If (p_1, q_1) and (p_2, q_2) are pairs of probability mass function, then*

$$D(\lambda p_1 + (1-\lambda)p_2||\lambda q_1 + (1-\lambda q_2)) \leqslant \lambda D(p_1||q_1) + (1-\lambda)D(p_2||q_2)$$

for all $0 \leqslant \lambda \leqslant 1$. That is, $D(p||q)$ is convex in the pair (p, q).

Further,

Theorem 2.2.12 *$H(p)$ is a concave function of p.*

Theorem 2.2.13 *Let $(X, Y) \sim p(x, y) = p(x)p(y|x)$. The mutual information $I(X;Y)$ is a concave function of $p(x)$ for fixed $p(y|x)$ and a convex function of $p(y|x)$ for fixed $p(x)$.*

The data processing inequality states that no matter what processing we perform on some data, we can not get more information out of a set of data than was there to begin with. In a sense, it provides a bound on how much can be accomplished with signal processing.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 2.2.6 *Random variable X, Y, and Z are said to form a Markov chain in that order, denoted by $X \to Y \to Z$ if the conditional distribution of Z depends only on Y and is independent of X. (That is, if we know Y, then knowing X also does not tell us any more than if we only know Y). if X, Y, and Z form a Markov chain, then the joint distribution can be written*

$$p(x, y, z) = p(x)p(y|x)p(z|y)$$

The concept of a state is that knowing the present state, the future of the system is independent of the past. The conditional independence idea means

$$p(x, z|y) = \frac{p(x, y, z)}{p(y)} = \frac{p(x, y)p(z|y)}{p(y)} = p(x|y)p(z|y)$$

Note, if $Z = f(Y)$ then $X \to Y \to Z$.

Theorem 2.2.14 *Data processing inequality*
If $X \to Y \to Z$, then

$$I(X; Y) \geqslant I(X; Z)$$

If we think of Z as being the result of some processing done on the data Y, that is, $Z = f(Y)$ for some deterministic or random function, then there is no function that can increase the amount of information that Y tells about X.

2.2.5 Feature selection

A fundamental problem of machine learning is to approximate the functional relationship $f(\cdot)$ between an input $X = \{x_1, x_2, .., x_M\}$ and an output Y, based on a memory of data points, $\{X_i, Y_i\}$, $i = 1, .., N$, where the X_i are vectors of reals and the Y_i are real numbers. There are some cases where the output Y is not determined by the complete set of the input features $X = \{x_1, x_2, .., x_M\}$, but it is only decided by a subset of them $\{x_{(1)}, x_{(2)}, .., x_{(m)}\}$, where $m < M$. When we have sufficient data and time, we can use all the input features, including the irrelevant ones, to approximate the underlying function between the input and the output. However, in practice the irrelevant features raise two problems in the learning process.

1. The irrelevant input features will induce greater computational cost.
2. The irrelevant input features may lead to overfitting

As a result, it is reasonable and important to ignore those input features with little effect on the output, so as to keep the size of the approximator model small. While the feature selection problem has been studied by the statistics and machine learning communities for decades, it has received much attention in the field of data mining. Some researchers call it filter models, while other classify it as wrapped around methods. It is also known as subset selection in the statistics community.

Since the computational cost of brute-force feature selection method is prohibitively high, with considerable danger of overfitting, people resort to greedy methods, such as forward selection. We must first clarify the problem of performance evaluation of a set of input features. Even if the feature sets are evaluated by test-set cross-validation or leave-one-out cross validation, an exhaustive search of possible feature sets is likely to find a misleadingly well scoring feature set by chance. To prevent this from happening, we can use a cascaded cross-validation procedure, which selects from increasingly large sets of features (and thus from increasingly large model classes).

1. Shuffle the data set and split into a training set of 70% of the data and a test-set of the remaining 30%.
2. Let j vary among feature set sizes $j = (0, 1, .., m)$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(a) Let f_{s_j} = best feature set of j, where best is measured as the minimiser of the leave-one-out cross-validation error over the training set.

(b) Let $Testscore_j$ = the RMS prediction error of feature set f_{s_j} on the test-set.

3. Select the feature set f_{s_j} for which the test-set score is minimised.

The score for the best feature set of a given size is computed by independent cross-validation from the score for the best size of feature set. Note, this procedure does not describe how the search for the best feature set of size j in step 2a) is done. Further, the performance evaluation of a feature selection algorithm is more complicated than the evaluation of the feature set. This is because we must first ask the algorithm to find the best feature subset. We must then give a fair estimate of how well the feature selection algorithm performs by trying the first step on different data sets. Hence, the full procedure of evaluating the performance of a feature selection algorithm (described above) should have two layers of loops (see algorithm below). The inner loop using an algorithm to find the best subset of features, and the outer loop evaluating the performance of the algorithm with different data sets.

1. Collect a training data set from the specific domain.
2. Shuffle the data set.
3. Break it into P partition, say $P = 20$.
4. For each partition ($i = 0, 1, .., P - 1$)

   (a) Let $OuterTrainset(i)$ = all partitions except i.

   (b) Let $OuterTestset(i)$ = the ith partition.

   (c) Let $InnerTrain(i)$ = randomly chosen 70% of the $OuterTrainset(i)$.

   (d) Let $InnerTest(i)$ = the remaining 30% of the $OuterTrainset(i)$.

   (e) For $j = 0, 1, .., m$
   Search for the best feature set with j components $f_{s_{ij}}$ using leave-one-out on $InnerTrain(i)$.
   Let $InnerTestScore_{ij}$ = RMS score of $f_{s_{ij}}$ on $InnerTest(i)$
   End loop of (j).

   (f) Select the $f_{s_{ij}}$ with the best inner test score.

   (g) Let $OuterScore_i$ = RMS score of the selected feature set on $OuterTestset(i)$.

   End of loop of (i).

5. Return the Mean Outer Score

We are now going to introduce the forward feature selection algorithm, and explore three greedy variants of this algorithm improving the computational efficiency without sacrificing too much accuracy.

The forward feature selection procedure begins by evaluating all feature subsets consisting of only one input attribute. We start by measuring the leave-one-out cross-validation (LOOCV) error of the one-component subsets $\{X_1\}, \{X_2\}, ..\{X_M\}$, where M is the input dimensionality, so that we can find the best individual feature $X_{(1)}$. Next, the forward selection finds the best subset consisting of two components, $X_{(1)}$, and one other feature from the remaining $M - 1$ input attributes. Hence, there are a total of $M - 1$ pairs. We let $X_{(2)}$ be the other attribute in the best pair besides $X_{(1)}$. Then, the input subsets with three, four, and more features are evaluated. According to the forward selection, the best subset with m features is the m-tuple consisting of $\{X_1\}, \{X_2\}, ..\{X_m\}$, while overall, the best feature set is the winner out of all the M steps. Assuming that the cost of a LOOCV evaluation with i features

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

is $C(i)$, then the computational cost of forward selection searching for a feature subset of size m out of M total input attributes will be

$$MC(1) + (M-1)C(2) + ... + (M-m+1)C(m)$$

For example, the cost of one prediction with one-nearest neighbour as the function approximator, using a kd-tree with j inputs, is $\mathcal{O}(j \log N)$ where N is the number of data points. Thus, the cost of computing the mean leave-one-out error, which involves N predictions, is $\mathcal{O}(jN \log N)$. Hence, the total cost of feature selection using the above formula is $\mathcal{O}(m^2 MN \log N)$.

An alternative solution to finding the total best input feature set is to employ exhaustive search. This method starts with searching the best one-component subset of the input features, which is the same in the forward selection algorithm. Then, we need to find the best two-component feature subset which may consist of any pairs of the input features. Afterwards, it consists in finding the best triple out of all the combinations of any three input features, etc. One can see that the cost of exhaustive search is

$$MC(1) + \binom{M}{2}C(2) + ... + \binom{M}{m}C(m)$$

and we see that the forward selection is much cheaper than the exhaustive search. However, the forward selection may suffer due to its greediness. For example, if $X_{(1)}$ is the best individual feature, it does not guarantee that either $\{X_{(1)}, X_{(2)}\}$ or $\{X_{(1)}, X_{(3)}\}$ must be better than $\{X_{(2)}, X_{(3)}\}$. As a result, a forward selection algorithm may select a feature set different from that selected by the exhaustive searching. Hence, with a bad selection of the input features, the prediction Y_q of a query $X_q = \{x_1, x_2, ..., x_M\}$ may be significantly different from the true Y_q.

In the case where the greediness of forward selection does not have a significantly negative effect on accuracy, we need to know how to modify the forward selection algorithm to be greedier in order to further improve the efficiency. There exists several greedier feature selection algorithms whose goal is to select no more than m features from a total of M input attributes, and with tolerable loss of prediction accuracy. We briefly discuss three of these algorithms

1. The super greedy algorithm: do all the 1-attribute LOOCV calculations, sort the individual features according to their LOOCV mean error, then take the m best features as the selected subset. Thus, we do M computations involving one feature and one computation involving m features. If the nearest neighbour is the function approximator, the cost of super greedy algorithm is $\mathcal{O}((M+m)N \log N)$.

2. The greedy algorithm: do all the 1-attribute LOOCV and sort them, take the best two individual features and evaluate their LOOCV error, then take the best three individual features, and so on, until m features have been evaluated. Compared with the super greedy algorithm, this algorithm may conclude at a subset whose size is smaller than m but whose inner tested error is smaller than that of the m component feature set. Hence, the greedy algorithm may end up with a better feature set than the super greedy one does. The cost of the greedy algorithm for the nearest neighbour is $\mathcal{O}((M+m^2)N \log N)$.

3. The restricted forward selection (RFS):

   (a) caculate all the 1-feature set LOOCV errors, and sort the features according to the corresponding LOOCV errors. We let the features ranking from the most important to the least important be $X_{(1)}, X_{(2)}, ..., X_{(M)}$.

   (b) do the LOOCV of 2-feature subsets consisting of the winner of the first round, $X_{(1)}$, along with another feature, either $X_{(2)}$ or $X_{(3)}$, or any other one until $X_{(\frac{M}{2})}$. There are $\frac{M}{2}$ of these pairs. The winner of this round will be the best 2-component feature subset chosen by RFS.

   (c) calculate the LOOCV errors of $\frac{M}{3}$ subsets consisting of the winner of the second round, along with the other $\frac{M}{3}$ features at the top of the remaining rank. In this way, the RFS will select its best feature triple.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(d) continue this procedure until RFS has found the best m-component feature set.

(e) From step 1) to step 4), the RFS has found m feature sets whose sizes range from 1 to m. By comparing their LOOCV errors, the RFS can find the best overall feature set.

One of the difference between RFS and conventional forward selection (FS) is that at each step, when inserting an additional feature into the subset, the FS considers all the remaining features, while the RFS only tries the part of them which seems the more promising. The cost of RFS for the nearest neighbour is $\mathcal{O}(MmN \log N)$.

We are left with finding out how cheap and how accurate all these varieties of forward selection are compared with the conventional forward selection method. Using real world data sets from StatLib/CMU and UCI's machine learning data repository coming from different domains such as biology, sociology, robotics etc, it was demonstrated that Exhaustive Search (ES) is prohibitively time consuming. Even though the features selected by FS may differ from the result of ES because some of the input features are not mutually independent, ES is far more expensive than the FS, while it is not significantly more accurate than FS. In order to investigate the influence of greediness on the above three greedy algorithms, we consider

1. the probabilities for these algorithms to select any useless features
2. the prediction errors using the feature set selected by these algorithms
3. the time cost for these algorithms to find their feature sets

It was found that FS does not eliminate more useless features than the greedier competitors except for the Super Greedy one. However, the greedier the algorithm, the more easily confused by the relevant but corrupted features it becomes. Further, the three greedier feature selection algorithms do not suffer great loss in accuracy, and RFS performs almost as well as the FS. As expected, the greedier algorithms improve the efficiency. It was found that the super greedy algorithm (Super) was ten time faster than the FS, while the greedy algorithm (Greedy) was seven times faster, and the restricted forward selection (RFS) was three times faster. Finally, the RFS performed better than the conventional FS in all aspects. Inserting more independent random noise and corrupted features to the data sets, the probability for any corrupted feature to be selected remained almost the same, while that of independent noise reduced greatly. To conclude, while in theory the greediness of feature selection algorithms may lead to great reduction in the accuracy of function approximation, in practice it does not often happen. The three greedier algorithms discussed above improve the efficiency of the forward selection algorithm, especially for larger data sets with high input dimensionalities, without significant loss in accuracy. Even in the case where the accuracy is more crucial than the efficiency, restricted forward selection is more competitive than the conventional forward selection.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 3

Data

3.1 Introduction to data

3.1.1 From data to information

We refer the readers to textbooks by Hand et al. [2001], Cormen et al. [2001], Allison [2001], Han et al. [2006], among others.

3.1.1.1 Data

3.1.1.1.1 Definitions Data is a set of values of qualitative or quantitative variables. There are three main types of data:

1. numeric: such as income, age (for a person)
2. categorical: such as gender, nationality
3. ordinal: such as low, medium, high.

In computing, data is information that has been translated into a form that is efficient for movement or processing. A digital computer represents a piece of data as a sequence of symbols drawn from a fixed alphabet. The most common digital computers use a binary alphabet, that is, an alphabet of two characters, typically denoted 0 and 1. Thus, data is information converted into binary digital form. It is acceptable for data to be used as a singular subject or a plural subject. More generally, data, information, knowledge and wisdom are closely related concepts, but each has its own role in relation to the other, and each term has its own meaning. Data is collected and analysed; data only becomes information suitable for making decisions once it has been analysed in some fashion. The concept of data in the context of computing was introduced by Claude Shannon who ushered in binary digital concepts based on applying two-value Boolean logic to electronic circuits (see Section (2.2)). The amount of information content in a data stream may be characterised by its Shannon entropy. Data is often assumed to be the least abstract concept, information the next least, and knowledge the most abstract. Thus, data becomes information by interpretation (see Ilkka [2000]). Generally, the concept of information is closely related to notions of constraint, communication, control, data, form, instruction, knowledge, meaning, mental stimulus, pattern, perception, and representation (see Beynon-Davies [2009]). This leads to the concept of data mining.

3.1.1.1.2 Dataset A dataset consists of feature vectors, where each feature vector is a description of an object by using a set of features. For example, in a three-Gaussian data set, a feature vector can be represented by $(value1, value2, cross)$ or $(value1, value2, circle)$, where $(value1, value2)$ are (x, y)-coordinates and $cross$ and

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$circle$ are labels. The number of features of a data set is called dimension. Features are also called attributes, a feature vector is also called an instance, and a data is sometime called a sample. Labelled data is a group of samples that have been tagged with one or more labels. Labelling typically takes a set of unlabelled data and augments each piece of that unlabelled data with meaningful tags that are informative. A data class refers to a class that contains only fields and crude methods for accessing them (getters and setters). These are simply containers for data used by other classes.

3.1.1.2 Features

3.1.1.2.1 Definition In machine learning and pattern recognition, a feature is an individual measurable property, or characteristic, of a phenomenon being observed (see Bishop [2006]). It is typically depicted by a column in a dataset. Inherent raw features are obtained directly from the dataset with no extra data manipulation or engineering, while derived features are usually obtained from feature engineering, where features are extracted from existing data attributes. Choosing informative, discriminating and independent features is a crucial step for effective algorithms in pattern recognition, classification and regression. Features are usually numeric, but structural features such as strings and graphs are used in syntactic pattern recognition. The concept of feature is related to that of explanatory variable used in statistical techniques such as linear regression.
For example, in character recognition, features may include histograms counting the number of black pixels along horizontal and vertical directions, number of internal holes, stroke detection and many others. In speech recognition, features for recognising phonemes can include noise ratios, length of sounds, relative power, filter matches and many others.

3.1.1.2.2 Feature engineering Feature engineering efforts have two main goals: (1) to prepare the proper input dataset compatible with the machine learning algorithm requirements (2) to improve the performance of machine learning models.
AI and ML models can only handle numeric features, so that both categorical and ordinal features must be converted into numeric features by creating dummy features. Transform a categorical feature into a set of dummy features, each representing a unique category. In that set, 1 indicates that the observation belong to that category.
Further, in the case of classification, continuous features have to be discretised. For instance, a set of numeric features can be conveniently described by a feature vector. As an example, one way of achieving binary classification is by using a linear predictor function (related to the perceptron described in Section (5.1.1)) with a feature vector as input. The method consists in calculating the scalar product between the feature vector and a vector of weights, comparing the result with a threshold, and deciding the class based on the comparison.
Some example of feature engineering techniques are:

1. Categorical data
   - One-hot encoding scheme: For a numeric representation of any categorical attribute with m labels, the scheme encodes, or transforms, the attribute into m binary features which can only contain a value of 1 or 0. Each observation in the categorical feature is thus converted into a vector of size m with only one of the values as 1 (indicating it is an active value.)
2. Numerical data
   - Binning
     - Fixed-width binning: It has fixed widths for each of the bins which are usually pre-defined by the user analysing the data. Each bin has a pre-fixed range of values which are assigned to that bin based on some domain knowledge, rules, or constraints.
     - Adaptive binning / quantile based binning: Quantiles are specific values or cut-points which help in partitioning the continuous valued distribution of a specific numeric field into discrete contiguous bins or intervals.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.1.2 Data representation

3.1.2.1 Units of information

The **bit** is a basic unit of information in computing and digital communications representing a logical state with one of two possible values. These values are most commonly represented as either 1 or 0, but other representations such as true-false, yes-no, $+/-$, or on-off are common. The correspondence between these values and the physical states of the underlying storage or device is a matter of convention, and different assignments may be used even within the same device or program. It may be physically implemented with a two-state device. A contiguous group of binary digits is commonly called a bit string, a bit vector, or a single-dimensional (or multi-dimensional) bit array. A group of eight binary digits is called **one byte**, but historically the size of the byte is not strictly defined. Frequently, half, full, double and quadruple words consist of a number of bytes which is a low power of two. In information theory (see Section (2.2)), one bit is the information entropy of a binary random variable that is 0 or 1 with equal probability, or the information that is gained when the value of such a variable becomes known. As a unit of information, the bit is also known as a shannon, named after Claude E. Shannon. Other units of information, sometimes used in information theory, include the natural digit also called a nat or nit and defined as $\log_2 e$ where e is the base of the natural logarithms. There is also the dit, ban, or hartley. These units of information are as follows:

- shannon or bit (base 2)
- nat (base e)
- trit (base 3)
- hartley, ban or dit (base 10)
- qubit (binary quantum)
- qutrit (ternary quantum)

Even though the bit is not defined in the International System of Units (SI), the International Electrotechnical Commission issued standard IEC 60027, which specifies that the symbol for binary digit should be bit, and this should be used in all multiples, such as kbit, for kilobit. However, the lower-case letter b is widely used as well and was recommended by the IEEE 1541 Standard (2002). In contrast, the upper case letter B is the standard and customary symbol for byte.
Multiple bits can be expressed and represented in several ways. For convenience of representing commonly reoccurring groups of bits in information technology, several units of information have traditionally been used, with the most common one being the byte, which historically was used to represent the group of bits used to encode a single character of text in a computer, until UTF-8 multibyte encoding took over. Nowadays, common implementation are using eight bits per byte. Computers usually manipulate bits in groups of a fixed size, conventionally named **words**. Like the byte, the number of bits in a word also varies with the hardware design, and is typically between 8 and 80 bits, or even more in some specialised computers. In the 21st century, retail personal or server computers have a word size of 32 or 64 bits.
The International System of Units defines a series of decimal prefixes for multiples of standardized units which are commonly also used with the bit and the byte. The prefixes kilo (10^3) through yotta (10^{24}) increment by multiples of 1000, and the corresponding units are the kilobit (kbit) through the yottabit (Ybit). This is illustrated in Figure (3.1).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Decimal				Binary					
Value		SI		Value		IEC		JEDEC	
1000	10^3	kbit	kilobit	1024	2^{10}	Kibit	kibibit	Kbit	kilobit
1000^2	10^6	Mbit	megabit	1024^2	2^{20}	Mibit	mebibit	Mbit	megabit
1000^3	10^9	Gbit	gigabit	1024^3	2^{30}	Gibit	gibibit	Gbit	gigabit
1000^4	10^{12}	Tbit	terabit	1024^4	2^{40}	Tibit	tebibit	-	
1000^5	10^{15}	Pbit	petabit	1024^5	2^{50}	Pibit	pebibit	-	
1000^6	10^{18}	Ebit	exabit	1024^6	2^{60}	Eibit	exbibit	-	
1000^7	10^{21}	Zbit	zettabit	1024^7	2^{70}	Zibit	zebibit	-	
1000^8	10^{24}	Ybit	yottabit	1024^8	2^{80}	Yibit	yobibit	-	

Figure 3.1: Multiples of bits.

3.1.2.2 Data structures

In computer science, a data structure is a data organisation, management, and storage format that enables efficient access and modification. That is, it is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data (see Wegner et al. [2003]). They serve as the basis for abstract data types (ADT), which defines the logical form of the data type. Data structures are based on the ability for a computer to fetch and store data at any place in its memory, specified by a pointer (a bit string), representing a memory address, that can be itself stored in memory and manipulated by the program. Thus, the array and record data structures are based on computing the addresses of data items with arithmetic operations, while the linked data structures are based on storing addresses of data items within the structure itself. The implementation of a data structure usually requires writing a set of procedures that create and manipulate instances of that structure.
There are numerous types of data structures, generally built upon simpler primitive data types:

- An array is a number of elements in a specific order, typically all of the same type, depending on the language, individual elements may either all be forced to be the same type, or may be of almost any type. Elements are accessed using an integer index to specify which element is required.
- A linked list (or list) is a linear collection of data elements of any type, called nodes, where each node has itself a value, and points to the next node in the linked list. The main advantage of a linked list over an array is that values can always be efficiently inserted and removed without relocating the rest of the list. Certain other operations, such as random access to a certain element, are however slower on lists than on arrays.
- A record (or tuple, struct) is an aggregate data structure. A record is a value that contains other values, typically in fixed number and sequence and typically indexed by names. The elements of records are usually called fields or members.
- A union is a data structure that specifies which of a number of permitted primitive types may be stored in its instances, such as float or long integer. Contrast with a record, which could be defined to contain a float and an integer; whereas in a union, there is only one value at a time. Enough space is allocated to contain the widest member datatype.
- A tagged union (or variant, variant record, discriminated union, disjoint union) contains an additional field indicating its current type, for enhanced type safety.
- An object is a data structure that contains data fields, like a record does, as well as various methods which operate on the data contents. An object is an in-memory instance of a class from a taxonomy. In the context of object-oriented programming, records are known as plain old data structures to distinguish them from objects.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, graphs and binary trees are other commonly used data structures.

3.1.2.2.1 Bit array A bit array (or bit map, bit set, bit string, bit vector) is an array of data structure that compactly stores bits. It can be used to implement a simple set of data structure. A bit array is effective at exploiting bit-level parallelism in hardware to perform operations quickly. A typical bit array stores kw bits, where w is the number of bits in the unit of storage, such as a byte or word, and k is some nonnegative integer. If w does not divide the number of bits to be stored, some space is wasted due to internal fragmentation.
More formally, a bit array is a mapping from some domain (almost always a range of integers) to values in the set $\{0, 1\}$. There are only two possible values, so that they can be stored in one bit. As with other arrays, the access to a single bit can be managed by applying an index to the array. Assuming its size (or length) to be n bits, the array can be used to specify a subset of the domain (such as $\{0, 1, 2, .., n-1\}$), where a 1-bit indicates the presence and a 0-bit the absence of a number in the set.
Each bit in a word can be singled out and manipulated using bitwise operations. Some examples include

- OR: It can be used to set a bit to one: 11101010 OR $00000100 = 11101110$
- AND: It can be used to set a bit to zero: 11101010 AND $11111101 = 11101000$
- AND together with zero-testing: It can be used to determine if a bit is set:

$$\begin{aligned} 11101010 \text{ AND } 00000001 &= 00000000 = 0 \\ 11101010 \text{ AND } 00000010 &= 00000010 \neq 0 \end{aligned}$$

- XOR: It can be used to invert or toggle a bit:

$$\begin{aligned} 11101010 \text{ XOR } 00000100 &= 11101110 \\ 11101110 \text{ XOR } 00000100 &= 11101010 \end{aligned}$$

- NOT: it can be used to invert all bits: NOT $10110010 = 01001101$

To obtain the bit mask needed for these operations, we can use a bit shift operator to shift the number 1 to the left by the appropriate number of places, as well as bitwise negation if necessary. Given two bit arrays of the same size representing sets, we can compute their union, intersection, and set-theoretic difference using n/w simple bit operations each ($2n/w$ for difference).

3.1.3 Character encoding

3.1.3.1 Character representation

In computing, character encoding is used to represent a repertoire of characters by some kind of encoding system that assigns a number to each character for digital representation. Depending on the abstraction level and context, corresponding code points and the resulting code space may be regarded as bit patterns, octets, natural numbers, electrical pulses, among others. A character encoding is used in computation, data storage, and transmission of textual data.
Formally, in character encoding terminology, a code point or code position is any of the numerical values that make up the codespace (the range of code points). The notion of a code point is used for abstraction, to distinguish both:

- the number from an encoding as a sequence of bits, and

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- the abstract character from a particular graphical representation (glyph).

Many code points represent single characters but they can also have other meanings, such as for formatting. For example, the character encoding scheme ASCII comprises 128 code points in the range 0_{hex} to $7F_{hex}$.
Some terminology related to character encoding are as follows:

- A character is a minimal unit of text that has semantic value.
- A character set is a collection of characters that might be used by multiple languages. Example: The Latin character set is used by English and most European languages, though the Greek character set is used only by the Greek language.
- A coded character set is a character set in which each character corresponds to a unique number.
- A code point of a coded character set is any allowed value in the character set or code space.
- A code space is a range of integers whose values are code points
- A code unit is a bit sequence used to encode each character of a repertoire within a given encoding form. This is referred to as a code value in some documents.

The character repertoire is an abstract set of more than one million characters found in a wide variety of scripts including Latin, Cyrillic, Chinese, Korean, Japanese, Hebrew, and Aramaic. Both the Unicode and GB18030 standards have a character repertoire. As new characters are added to one standard, the other standard also adds those characters, to maintain parity. The code unit size is equivalent to the bit measurement for the particular encoding:

- A code unit in US-ASCII consists of 7 bits;
- A code unit in UTF-8, EBCDIC and GB18030 consists of 8 bits;
- A code unit in UTF-16 consists of 16 bits;
- A code unit in UTF-32 consists of 32 bits.

3.1.3.2 Unicode encoding

Unicode is an information technology standard for the consistent encoding, representation, and handling of text expressed in most of the writing systems in the world. Rather than mapping characters directly to octets (bytes), they separately define what characters are available, corresponding natural numbers (code points), how those numbers are encoded as a series of fixed-size natural numbers (code units), and finally how those units are encoded as a stream of octets. The purpose of this decomposition is to establish a universal set of characters that can be encoded in a variety of ways. Practically, the Unicode Standard consists of a set of code charts for visual reference, an encoding method and set of standard character encodings, a set of reference data files, and a number of related items, such as character properties, rules for normalisation, decomposition, collation, rendering, and bidirectional text display order.
Unicode can be implemented by different character encodings. The Unicode standard defines Unicode Transformation Formats (UTF) UTF-8, UTF-16, and UTF-32, and several other encodings. The most commonly used encodings are UTF-8 (dominant encoding on the World Wide Web), UTF-16, and UCS-2. The UTF-8 uses one byte for the first 128 code points, and up to 4 bytes for other characters.
Unicode equivalence is the specification by the Unicode character encoding standard that some sequences of code points represent essentially the same character. It allows compatibility with pre-existing standard character sets, which often included similar or identical characters. Unicode provides two such notions:

1. Canonical equivalence: Code point sequences that are defined as canonically equivalent are assumed to have the same appearance and meaning when printed or displayed.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. Canonical compatibility: Sequences that are defined as compatible are assumed to have possibly distinct appearances, but the same meaning in some contexts.

The implementation of Unicode string searches and comparisons in text processing software must take into account the presence of equivalent code points. In the absence of this feature, users searching for a particular code point sequence would be unable to find other visually indistinguishable glyphs that have a different, but canonically equivalent, code point representation. Unicode normalisation provides standard normalisation algorithms that replaces equivalent sequences of characters so that any two texts that are equivalent will be reduced to the same sequence of code points, called the normalisation form, or normal form, of the original text. It produces a unique (normal) code point sequence for all sequences that are equivalent. The equivalence criteria can be either canonical (NF) or compatibility (NFK). Unicode provides two normal forms that are semantically meaningful for each of the two compatibility criteria:

1. the composed forms:
   - NFC (Normalisation Form Canonical Composition): Characters are decomposed and then recomposed by canonical equivalence.
   - NFKC (Normalization Form Compatibility Composition): Characters are decomposed by compatibility, then recomposed by canonical equivalence.
2. the decomposed forms
   - NFD (Normalisation Form Canonical Decomposition): Characters are decomposed by canonical equivalence, and multiple combining characters are arranged in a specific order.
   - NFKD (Normalisation Form Compatibility Decomposition): Characters are decomposed by compatibility, and multiple combining characters are arranged in a specific order.

Both the composed and decomposed forms impose a canonical ordering on the code point sequence, which is necessary for the normal forms to be unique. The canonical ordering is mainly concerned with the ordering of a sequence of combining characters.
In order to compare or search Unicode strings, software can use either the composed or decomposed forms. This choice does not matter as long as it is the same for all strings involved in a search, comparison. On the other hand, the choice of equivalence criteria can affect search results. When two applications share Unicode data, but normalise them differently, it can results in errors and data loss.

3.1.3.3 One-hot encoding

In digital circuits and machine learning, a one-hot is a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0) (see Harris et al. [2012]). In statistics, dummy variables represent a similar technique for representing categorical data. One-hot encoding is often used for indicating the state of a state machine. For instance, when using binary or Gray code, a decoder is needed to determine the state. However, a one-hot state machine, does not need a decoder as the state machine is in the nth state if and only if the nth bit is high. This is illustrated in Figure (3.2).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Binary	Gray code	One-hot
000	000	00000001
001	001	00000010
010	011	00000100
011	010	00001000
100	110	00010000
101	111	00100000
110	101	01000000
111	100	10000000

Figure 3.2: One-hot table.

In natural language processing, a one-hot vector is a $1 \times N$ matrix (vector) used to distinguish each word in a vocabulary from every other word in the vocabulary. The vector consists of 0s in all cells with the exception of a single 1 in a cell used uniquely to identify the word. This ensures that machine learning does not assume that higher numbers are more important. For example, the value 8 is bigger than the value 1, but that does not make 8 more important than 1. The same is true for words: the value **New York** is not more important than **York**. In machine learning, one-hot encoding is a frequently used method to deal with categorical data. Since ML models need their input variables to be numeric, categorical variables must be transformed in the pre-processing part. Categorical data is of two types:

- Nominal: There is no quantitative relationship between nominal variables.
- Ordinal: It has a ranked order for its values and can therefore be converted to numerical data through ordinal encoding. For example, the ratings on a test ranging from A to F, which could be ranked using numbers from 1 to 6.

Sine we can not use ordinal encoding on nominal variables, one-hot encoding is often applied to nominal variables, in order to improve the performance of the algorithm. For each unique value in the original categorical column, a new column is created in this method. These dummy variables are then filled up with zeros and ones (1 meaning TRUE, 0 meaning FALSE). This is illustrated in Figure (3.3).

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Figure 3.3: One-hot encoding example.

Note, as this process creates multiple new variables, it is prone to creating too many predictors if there are many unique values in the original column. Further, it causes multicollinearity between the individual variables, which potentially reduces the model's accuracy.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.2 Data storage

3.2.1 Definitions

3.2.1.1 Database model

A data base model (db-model) is a collection of conceptual tools used to model representations of real-world entities and the relationships among them (see Silberschatz et al. [1996]). The term is also often used to refer to a collection of data structure types. There are many factors influencing the evolution and diversity of db-models. Clearly, each db-model proposal is based on certain theoretical principles, and serves as base for the development of related models. Mendelzon et al. [1989] presented a graph for the evolution of database models which we display in Figure (3.4).

Figure 3.4: Evolution of database models. Rectangles denote database models, arrows indicate influences, and circles denote theoretical developments.

We will distinguish data as

1. data in a single table, without multi-valued attributes.
2. tree data, such as data warehouse hierarchies or XML. data.
3. data represented as a graph (data for personal information management).

See textbooks by Kroenke et al. [2007], Liu et al. [2009b], Silberschatz [2010].

3.2.1.2 Database

A database is an organised collection of data, often developed using formal design and modelling techniques, generally stored and accessed electronically from a computer system. Access to this data is usually provided by a database

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

management system (DBMS), which is the software (1) interacting with end users, applications, and the database itself to capture and analyse the data as well as (2) encompassing the core facilities provided to administer the database. This is illustrated in Figure (3.5). It consists of an integrated set of computer software that allows users to interact with one or more databases and provides access to all of the data contained in the database (although restrictions may exist that limit access to particular data). It enables users to define, create, maintain and control access to the database. Examples include MySQL, PostgreSQL, MSSQL, Oracle Database, and Microsoft Access. MySQL is an open-source RDBMS working with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

Figure 3.5: DBMS flow.

DBMS provide various functions that allow entry, storage and retrieval of large quantities of information and provides ways to manage how that information is organised. The functions allowing for the management of a database and its data can be classified into four main functional groups:

- Data definition: Creation, modification and removal of definitions that define the organisation of the data.
- Update: Insertion, modification, and deletion of the actual data.
- Retrieval: Providing information in a form directly usable or for further processing by other applications. The retrieved data may be made available in a form basically the same as it is stored in the database or in a new form obtained by altering or combining existing data from the database.
- Administration: Registering and monitoring users, enforcing data security, monitoring performance, maintaining data integrity, dealing with concurrency control, and recovering information that has been corrupted by some event such as an unexpected system failure.

Both a database and its DBMS conform to the principles of a particular database model. A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organised, and manipulated. The most popular example of a database model is the relational model (or the SQL approximation of relational), which uses a table-based format. In general, database system (DS) refers collectively to the database model, database management system, and database. Databases and DBMSs can be categorised according to the database model(s) that they support (such as relational or XML), the type(s) of computer they run on (from a server cluster to a mobile phone), the query language(s) used to access the database (such as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

SQL or XQuery), and their internal engineering, which affects performance, scalability, resilience, and security. SQL (Structured Query Language) both introduces the concept of accessing many records with one single command, and eliminates the need to specify how to reach a record, e.g. with or without an index. The scope of SQL includes data query, data manipulation (insert, update and delete), data definition (schema creation and modification), and data access control.

Computer scientists may classify database-management systems according to the database models that they support. Dominant in the 1980s, relational databases (RDBs) model data as rows and columns in a series of tables, and the vast majority use SQL (Structured Query Language) for writing and querying data. The 1990s, along with a rise in object-oriented programming, saw a growth in how data in various databases were handled. Programmers and designers began to treat the data in their databases as objects, allowing for relations between data to be relations to objects and their attributes and not to individual fields. That is, if a person's data were in a database, that person's attributes, such as their address, phone number, and age, were now considered to belong to that person instead of being extraneous data. Object databases and object-relational databases attempt to solve the object-relational impedance mismatch (describing the inconvenience of translating between programmed objects and database tables) by providing an object-oriented language (sometimes as extensions to SQL) that programmers can use as alternative to purely relational SQL. In the 2000s, non-relational databases became popular, referred to as NoSQL because they use different query languages. They introduced fast key-value stores and document-oriented databases. NoSQL databases are often very fast, do not require fixed table schemas, avoid join operations by storing denormalized data, and are designed to scale horizontally. These databases are increasingly used in big data and real-time web applications. Some reasons for using this approach include: simplicity of design, simpler horizontal scaling to clusters of machines (which is a problem for relational databases), finer control over availability and limiting the object-relational impedance mismatch. The data structures used by NoSQL databases (such as key-value pair, wide column, graph, or document) are different from those used by default in relational databases, making some operations faster in NoSQL. However, few systems maintain both ACID transactions and X/Open XA standards for distributed transaction processing. There are various ways to classify NoSQL databases, with different categories and subcategories, some of which overlap. A basic classification by data model is:

- Wide column store: It uses tables, rows, and columns, but unlike a relational database, the names and format of the columns can vary from row to row in the same table. A wide-column store can be interpreted as a two-dimensional key-value store.
- Document stores: They assume that documents encapsulate and encode data (or information) in some standard formats or encodings. Encodings in use include XML, YAML, and JSON and binary forms like BSON. Documents are addressed in the database via a unique key that represents that document. Another defining characteristic of a document-oriented database is an API or query language to retrieve documents based on their contents. XML databases are a type of structured document-oriented database that allows querying based on XML document attributes. They are mostly used in applications where the data is conveniently viewed as a collection of documents, with a structure that can vary from the very flexible to the highly rigid.
- Key-value stores (KV): They use the associative array (also called a map or dictionary) as their fundamental data model. In this model, data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection. They can use consistency models ranging from eventual consistency to serializability. Some databases support ordering of keys. There are various hardware implementations, and some users store data in memory (RAM), while others on solid-state drives (SSD) or rotating disks (aka hard disk drive (HDD)).
- Graph databases: They are designed for data whose relations are well represented as a graph consisting of elements connected by a finite number of relations. Examples of data include social relations, public transport links, road maps, network topologies, etc.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Object databases: They are object-oriented database management systems (OODBMS) in which information is represented in the form of objects as used in object-oriented programming. They allow object-oriented programmers to develop the product, store them as objects, and replicate or modify existing objects to make new objects within the OODBMS. Since the database is integrated with the programming language, the programmer can maintain consistency within one environment, in that both the OODBMS and the programming language will use the same model of representation. Relational DBMS projects, by way of contrast, maintain a clearer division between the database model and the application. Object-relational databases are a hybrid of both objects and relational approaches. As the usage of web-based technology increases with the implementation of Intranets and extranets, companies have a vested interest in OODBMSs to display their complex data.

3.2.2 Relational database

3.2.2.1 Description

A relational database (RDB) is a digital database based on the relational model of data (see Codd [1970]), and a software system used to maintain relational databases is a relational database management system (RDBMS). Many relational database systems have an option of using the SQL (Structured Query Language) for querying and maintaining the database. RDB organises data into one or more tables (or relations) of columns and rows, with a unique key identifying each row. Rows are also called records or tuples, and columns are also called attributes. Relational database terms and corresponding SQL term are displayed in Table (3.1). Generally, each table/relation represents one entity type (such as customer or product). The rows represent instances of that type of entity (such as Lee or chair) and the columns representing values attributed to that instance (such as address or price).

SQL term	Relational database term	Description
Row	Tuple or record	A data set representing a single item
Column	Attribute or field	A labelled element of a tuple, such as Address or Date of birth
Table	Relation or base relvar	A set of tuples sharing the same attributes: a set of columns and rows
View or result set	Derived relvar	Any set of tuples: a data report from the RDBMS in response to a query

Table 3.1: Relational database terms and corresponding SQL term.

Objects are typically physical objects or concepts. A relation is defined as a set of tuples that have the same attributes, where the tuple represents an object and information about that object. A relation is usually described as a two-dimensional table, which is organised into rows and columns. All the data referenced by an attribute are in the same domain and conform to the same constraints. This is illustrated in Figure (3.6).

Figure 3.6: Relational DB terminology.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Relationships are a logical connection between different tables, established on the basis of interaction among these tables. The relational model specifies that the tuples of a relation have no specific order and that the tuples, in turn, impose no order on the attributes. Applications access data by specifying queries, which use operations such as

- select to identify tuples
- project to identify attributes
- join to combine relations.

Relations can be modified using the insert, delete, and update operators. New tuples can supply explicit values or be derived from a query. Similarly, queries identify tuples for updating or deleting.

A **domain** describes the set of possible values for a given attribute, and can be considered a constraint on the value of the attribute. Constraints make it possible to further restrict the domain of an attribute. For instance, a constraint can restrict a given integer attribute to values between 1 and 10. Constraints provide one method of implementing business rules in the database and support subsequent data use within the application layer. Constraints restrict the data that can be stored in relations, and are usually defined using expressions resulting in a boolean value, indicating whether or not the data satisfies the constraint. Constraints can apply to single attributes, to a tuple (restricting combinations of attributes) or to an entire relation. Since every attribute has an associated domain, they are domain constraints. The two principal rules for the relational model are known as

1. entity integrity:
2. referential integrity: It is based on the simple concept of relational vector based analytic algorithms. It is based on the simple concept of relational vector based analytic algorithms.

Each row (or tuple) in a table (see Figure (3.6)) has its own unique key, so that rows in a table can be linked to rows in other tables by adding a column for the unique key of the linked row. Part of this processing involves consistently being able to select or modify one and only one row in a table. Therefore, most physical implementations have a unique primary key (PK) for each row in a table used to define the relationships among the tables. In addition, a surrogate key is an artificial attribute assigned to an object which uniquely identifies it (for instance, in a table of information about students at a school they might all be assigned a student ID in order to differentiate them). It has no intrinsic (inherent) meaning, but rather is useful through its ability to uniquely identify a tuple. Another common occurrence, especially in regard to N:M cardinality is the composite key, which is a key made up of two or more attributes within a table that (together) uniquely identify a record.
When a new row is written to the table, a new unique value for the primary key is generated: this is the key that the system uses primarily for accessing the table. When a PK migrates to another table, it becomes a foreign key in the other table. System performance is optimised for PKs. Other, more natural keys may also be identified and defined as alternate keys (AK). Often several columns are needed to form an AK (this is one reason why a single integer column is usually made the PK). Both PKs and AKs have the ability to uniquely identify a row within a table. Additional technology may be applied to ensure a unique ID across the world, a globally unique identifier, when there are broader system requirements.
A simple way of providing quick access to data is to use **indices** which can be created on any combination of attributes on a relation. Queries that filter using those attributes can find matching tuples directly using the index (similar to Hash table lookup), without having to check each tuple in turn. This is analogous to using the index of a book to go directly to the page on which the information you are looking for is found, so that you do not have to read the entire book to find what you are looking for. Relational databases typically supply multiple indexing techniques, each of which being optimal for some combination of data distribution, relation size, and typical access pattern. They can be implemented via B+ trees, R-trees, and bitmaps.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.2.2.2 Database normalisation

Database normalisation is the process of structuring a relational database in accordance with a series of normal forms (NF) in order to reduce data redundancy and improve data integrity. It entails organising the columns (attributes) and tables (relations) of a database to ensure that their dependencies are properly enforced by database integrity constraints. It encompasses a set of procedures designed to eliminate non-simple domains (non-atomic values) and the redundancy (duplication) of data, which in turn prevents data manipulation anomalies and loss of data integrity (it supports ACID transactions). The most common forms of normalisation applied to databases to reduce data redundancy, preserve data consistency, and improve data integrity are called the normal forms. It entails organising the columns (attributes) and tables (relations) of a database to ensure that their dependencies are properly enforced by database integrity constraints. This is accomplished by applying some formal rules either by a process of synthesis (creating a new database design) or decomposition (improving an existing database design). A fully normalised database allows its structure to be extended to accommodate new types of data without changing existing structure too much. As a result, applications interacting with the database are minimally affected (see Kent [1983]). The process is progressive, and a higher level of database normalization cannot be achieved unless the previous levels have been satisfied. The main objective of the first normal form (1NF) is to permit data to be queried and manipulated using a universal data sub-language grounded in first-order logic (SQL is an example of such a data sub-language). Informally, a relational database relation is often described as normalised if it meets third normal form (3NF) (see Codd [1971]). The objectives of normalisation beyond 1NF are stated as follows:

- To free the collection of relations from undesirable insertion, update and deletion dependencies.
- To reduce the need for restructuring the collection of relations, as new types of data are introduced, and thus increase the life span of application programs.
- To make the relational model more informative to users.
- To make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.

When modifying (update, insert into, or delete from) a relation, the following undesirable side-effects may arise in relations that have not been sufficiently normalised:

- Update anomaly: The same information can be expressed on multiple rows, so that updates to the relation may result in logical inconsistencies. For example, each record in an "Employees' Skills" relation might contain an Employee ID, Employee Address, and Skill. Thus a change of address for a particular employee may need to be applied to multiple records (one for each skill). If the update is only partially successful, the employee's address is updated on some records but not others, then the relation is left in an inconsistent state. Specifically, the relation provides conflicting answers to the question of what this particular employee's address is.
- Insertion anomaly: There are circumstances in which certain facts cannot be recorded at all. For example, each record in a "Faculty and Their Courses" relation might contain a Faculty ID, Faculty Name, Faculty Hire Date, and Course Code. Therefore, we can record the details of any faculty member who teaches at least one course, but we cannot record a newly hired faculty member who has not yet been assigned to teach any courses, except by setting the Course Code to null.
- Deletion anomaly: Under certain circumstances, deletion of data representing certain facts necessitates deletion of data representing completely different facts. The "Faculty and Their Courses" relation described in the previous example suffers from this type of anomaly, for if a faculty member temporarily ceases to be assigned to any courses, we must delete the last of the records on which that faculty member appears, effectively also deleting the faculty member, unless we set the Course Code to null.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.2.2.3 Storing tables

The most expensive operations involving hard disks are seeks. Thus, in order to improve overall performance, related data should be stored in a way that minimises the number of seeks. This is known as locality of reference. A common method of storing a table is to serialise each row of data. As data is inserted into the table, it is assigned an internal ID, the *rowid* that is used internally in the system to refer to data. In this case the records have sequential *rowids* independent of the user-assigned *empid*. Row-oriented systems are designed to efficiently return data for an entire row, or record, in as few operations as possible. By storing the record's data in a single block on the disk, along with related records, the system can quickly retrieve records with a minimum of disk operations. However, row-oriented systems are not efficient at performing set-wide operations on the whole table, as opposed to a small number of specific records. To improve the performance of these sorts of operations (which are very common, and generally the point of using a DBMS), most DBMSs support the use of database indexes, which store all the values from a set of columns along with *rowid* pointers back into the original table.
As they store only single pieces of data, rather than entire rows, indexes are generally much smaller than the main table stores. Scanning this smaller set of data reduces the number of disk operations. If the index is heavily used, it can dramatically reduce the time for common operations. However, maintaining indexes adds overhead to the system, especially when new data is written to the database. Records not only need to be stored in the main table, but any attached indexes have to be updated as well. The main reason why indexes dramatically improve performance on large datasets is that database indexes on one or more columns are typically sorted by value, which makes range queries operations very fast (lower time-complexity).
Note, a number of row-oriented databases have been designed to fit entirely in RAM, an in-memory database. These systems do not depend on disk operations, and have equal-time access to the entire dataset, reducing the need for indexes, as it requires the same amount of operations to fully scan the original data as a complete index for typical aggregation purposes. Such systems may be therefore simpler and smaller, but can only manage databases that will fit in memory.

3.2.2.4 Column-oriented DBMS

A column-oriented DBMS is a database management system (DBMS) storing data tables by column rather than by row as in the above relational DBMS. This is illustrated in Figure (3.7). Practical use of a column store versus a row store differs little in the relational DBMS world: Both columnar and row databases can use traditional database query languages like SQL to load data and perform queries. Both row and columnar databases can become the backbone in a system to serve data for common extract, transform, load (ETL) and data visualisation tools (see Section (3.4)). However, by storing data in columns rather than in rows, the database can more precisely access the data it needs to answer a query rather than scanning and discarding unwanted data in rows.

Figure 3.7: Row versus column oriented data base.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

A column-oriented database serialises all of the values of a column together, then the values of the next column, and so on. Even though a column-oriented store is similar to a row-store with an index on every column, the mapping of the data that differs dramatically. That is, in a row-oriented indexed system, the primary key is the *rowid* that is mapped from indexed data, while in the column-oriented system, the primary key is the data, which is mapped from *rowids*. Whether or not a column-oriented system will be more efficient in operation depends heavily on the workload being automated. Operations that retrieve all the data for a given object (the entire row) are slower. A row-oriented system can retrieve the row in a single disk read, whereas numerous disk operations to collect data from multiple columns are required from a columnar database. However, these whole-row operations are generally rare. Comparisons between row-oriented and column-oriented databases are typically concerned with the efficiency of hard-disk access for a given workload, as seek time is incredibly long compared to the other bottlenecks in computers. Clearly, disk access is a major bottleneck in handling big data. Columnar databases boost performance by reducing the amount of data that needs to be read from disk, both by efficiently compressing the similar columnar data and by reading only the data necessary to answer the query. In practice, columnar databases are well-suited for OLAP-like workloads (such as data warehouses) which typically involve highly complex queries over all data (possibly petabytes), but some work must be done to write data into a columnar database.

As an example, kdb+ is a column-based relational time series database (TSDB) with in-memory (IMDB) abilities, developed and marketed by Kx Systems. The database is commonly used in high-frequency trading to store, analyse, process, and retrieve large data sets at high speed. Financial institutions use kdb+ to analyse time series data such as stock or commodity exchange data. Commonly accessed data is pushed into random-access memory (RAM), which is faster to access than data in disk storage. Created with financial institutions in mind, the database was developed as a central repository to store time series data that supports real-time analysis of billions of records. kdb+ has the ability to analyse data over time and responds to queries similar to Structured Query Language (SQL).

3.2.3 Graph database

3.2.3.1 Description

Graph database models can be defined as those in which data structures for the schema and instances are modelled as graphs or generalisations of them, and data manipulation is expressed by graph-oriented operations and type constructors (see Section (13.1) for graph theory). A graph database (GDB) is a database that uses graph structures for semantic queries with nodes, edges, and properties to represent and store data. The graph relates the data items in the store to a collection of nodes and edges, the edges representing the relationships between the nodes (see Bourbakis [1998]). The GDP consists of a set of objects:

- Nodes: They represent entities or instances such as people, businesses, accounts, or any other item to be tracked. They are roughly the equivalent of a record, relation, or row in a relational database, or a document in a document-store database.
- Edges (or relationships): They are the lines that connect nodes to other nodes, representing the relationship between them. Meaningful patterns emerge when examining the connections and interconnections of nodes, properties and edges. The edges can either be directed or undirected. In an undirected graph, an edge connecting two nodes has a single meaning. In a directed graph, the edges connecting two different nodes have different meanings, depending on their direction. Edges are the key concept in graph databases, representing an abstraction that is not directly implemented in a relational model or a document-store model.
- Properties: They are information associated to nodes. For example, if Wikipedia were one of the nodes, it might be tied to properties such as website, reference material, or words that starts with the letter w, depending on which aspects of Wikipedia are germane to a given database.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

These relationships allow data in the store to be linked together directly and, in many cases, retrieved with one operation. Graph databases hold the relationships between data as a priority, making any query fast and efficient. A simple relationship for emails exchange is illustrated in Figure (3.8).

Figure 3.8: Graph relationship for emails exchange.

Graph databases are a powerful tool for graph-like queries such as computing the shortest path between two nodes in the graph. The relative advantage of graph retrieval grows with the complexity of a query. Recently, the need to manage information with graph-like nature has re-established the relevance of this area. In a survey, Angles et al. [2008] presented the work that has been conducted in the area of graph database modelling, concentrating on data structures, query languages, and integrity constraints.

Some GDBs depend on a relational engine to store the data in a table, while others use a key-value store, or document-oriented database for storage, making them inherently NoSQL structures. That is, a node can be represented as any other document store, but edges that link two different nodes hold special attributes inside its document (a **from** and **to** attributes). Thus, graph databases are a type of NoSQL database, created to address the limitations of relational databases. This is because in the former the model explicitly lays out the dependencies between nodes of data, while in the latter the model link the data by implicit connections. The relationships in the database can be labelled, directed, and given properties. GDBs are often faster for associative data sets and map more directly to the structure of object-oriented applications. They can scale more naturally to large datasets as they do not typically need join operations, which can often be expensive. As they depend less on a rigid schema, they are marketed as more suitable to manage ad hoc and changing data with evolving schemas.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

As discussed in the above section, relational models require a strict schema and data normalisation imposed limitations on how relationships can be queried. Even though relational models achieve a fast row-by-row access, problems arise with when there is a need to form complex relationships between the stored data. Although relationships can be analysed with the relational model, complex queries performing many join operations on many different attributes over several tables are required. In working with relational models, foreign key constraints should also be considered when retrieving relationships, causing additional overhead. On the other hand, graph databases are often faster for associative data sets and map more directly to the structure of object-oriented applications. They can scale more naturally to large datasets as they do not typically need join operations, which can often be expensive. As they depend less on a rigid schema, they are marketed as more suitable to manage ad hoc and changing data with evolving schemas.

There are multiple types of graphs that can be categorised. Some of the main categories of graphs are:

- Social graph: It represents the connections between people. Examples include Facebook, Twitter, and the idea of six degrees of separation.
- Intent graph: It deals with reasoning and motivation.
- Consumption graph (also known as the payment graph): It is heavily used in the retail industry. E-commerce companies such as Amazon, eBay and Walmart use consumption graphs to track the consumption of individual customers.
- Interest graph: It maps a person's interests and is often complemented by a social graph. It has the potential to follow the previous revolution of web organisation by mapping the web by interest rather than indexing webpages.
- Mobile graph: It is built from mobile data. Mobile data in the future may include data from the web, applications, digital wallets, GPS, and Internet of Things (IoT) devices.

In order to retrieve data from a graph database, one requires a query language other than SQL, which was designed for the manipulation of data in a relational system. Originally, no universal graph query language was adopted, resulting a wide variety of systems, most often tightly tied to one product. Some standardisation efforts have occurred, leading to multi-vendor query languages like Gremlin, SPARQL, and Cypher. In addition to having query language interfaces, some graph databases are accessed through application programming interfaces (APIs).

3.2.3.2 Relational versus graph database

As discussed above, in a traditional relational or SQL database, the data is organised into tables. Each table records data in a specific format with a fixed number of columns, each column with its own data type (integer, time/date, freeform text, etc.). This model works best when you are dealing mainly with data from any one table. It also does not work too badly when you are aggregating data stored across multiple tables. However, that behaviour has some notable limits. For example, consider a music database, with albums, bands, labels, and performers. If you want to report all the performers that were featured on **this** album by **that** band released on **these** labels, that is, four different tables, you have to explicitly describe those relationships. With a relational database, this is accomplished by way of new data columns (for one-to-one or one-to-many relationships), or new tables (for many-to-many relationships).

This is only practical as long as you are managing a modest number of relationships. If you are dealing with millions or even billions of relationships, such as friends of friends of friends, those queries do not scale well. In short, if the relationships between data, not the data itself, are your main concern, then a different kind of database, such as a graph database, is in order. In a conventional database, queries about relationships can take a long time to process. This is because relationships are implemented with foreign keys and queried by joining tables. However, performing joins is expensive, especially when you must sort through large numbers of objects or, worse, when you must join multiple tables to perform the sorts of indirect (such as friend of a friend) queries that graph databases excel at.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Graph databases work by storing the relationships along with the data. Since related nodes are physically linked in the database, accessing those relationships is as immediate as accessing the data itself. That is, instead of **calculating** the relationship as relational databases must do, graph databases simply **read** the relationship from storage. Thus, satisfying queries is a simple matter of walking, or, traversing the graph. Traversals in graph databases do not require computationally expensive joins and creation of indexes. Further, a graph database not only stores the relationships between objects in a native way, making queries about relationships fast and easy, but allows you to include different kinds of objects and different kinds of relationships in the graph. Due to the fact that graph databases are schema-free, adding new data to them is simply done by adding new nodes and the relationships between them. That means that the pre-existing relationships will not be affected, and the querying time will not change significantly. Thus, in terms of performance and flexibility, graph databases conform closer to document databases or key-value stores than do relational or table-oriented databases.

Graph Databases address some of the key challenges experienced with linked datasets, such as:

1. The changes of the linked data over time when there are data amendments, or, when additional data sources arrive and time-points are considered. Any updates may create conflicts in the link status.
2. Different users may require different quality of links (even when linking the same datasets)
3. The need to clerically resolve only the difficult cases (target the clerical resource into the most uncertain links)
4. The lack of triangulation of linkage issue which is a result of contradictions in the link status within groups of records between multiple different datasets
5. The long data load, querying time and amount of storage memory required
6. The privacy and confidentiality concerns for both the data custodian and the linkage unit

3.2.4 Big data

3.2.4.1 Definitions

Big data includes data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process data within a tolerable elapsed time. It is a field that treats ways to analyse, systematically extract information from, or otherwise deal with data sets that are too large or complex to be dealt with by traditional data-processing application software. It uses mathematical analysis, optimisation, inductive statistics and concepts from nonlinear system identification to infer laws (regressions, nonlinear relationships, and causal effects) from large sets of data with low information density to reveal relationships and dependencies, or to perform predictions of outcomes and behaviours. Nowadays, big data refers to the use of predictive analytics, user behaviour analytics, or certain other advanced data analytics methods that extract value from data, and seldom to a particular size of data set. Some of the challenges of big data include capturing data, data storage, data analysis, search, sharing, transfer, visualisation, querying, updating, information privacy and data source. It is often characterised by five Vs:

- Volume: It is the quantity of generated and stored data. The size of the data determines the value and potential insight, and whether it can be considered big data or not. The size of big data is usually larger than terabytes and petabytes.
- Velocity: The speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development. Big data is often available in real-time. Compared to small data, big data is produced more continually. There are two kinds of velocity related to big data: (1) the frequency of generation, and (2) the frequency of handling, recording, and publishing.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Variety: It is the type and nature of the data. It can be semi-structured, unstructured, and structured of disparate types. New technologies aim at capturing, storing, and processing the semi-structured and unstructured (variety) data generated with high speed(velocity), and huge in size (volume). Big data draws from text, images, audio, video; plus it completes missing pieces through data fusion.
- Veracity: It is the extended definition for big data, which refers to the data quality and the data value.
- Value: The utility that can be extracted from the data.

The Zettabyte Era is a period of human and computer science history that started in the mid-2010s. The precise starting date depends on whether it is defined as when the global IP traffic first exceeded that of one zettabyte, which happened in 2016, or when the amount of digital data in the world first exceeded a zettabyte, which happened in 2012. A zettabyte is a multiple of the unit byte that measures digital storage, and it is equivalent to $1,000,000,000,000,000,000,000$ (or 10^{21}) bytes [1]. The order of magnitude of data is displayed in Figure (3.9).

Multiple-byte units V·T·E						
Decimal			Binary			
Value	Metric		Value	IEC		JEDEC	
1000	kB	kilobyte	1024	KiB	kibibyte	KB	kilobyte
1000^2	MB	megabyte	1024^2	MiB	mebibyte	MB	megabyte
1000^3	GB	gigabyte	1024^3	GiB	gibibyte	GB	gigabyte
1000^4	TB	terabyte	1024^4	TiB	tebibyte	–	
1000^5	PB	petabyte	1024^5	PiB	pebibyte	–	
1000^6	EB	exabyte	1024^6	EiB	exbibyte	–	
1000^7	ZB	zettabyte	1024^7	ZiB	zebibyte	–	
1000^8	YB	yottabyte	1024^8	YiB	yobibyte	–	
Orders of magnitude of data							

Figure 3.9: The order of magnitude of data.

Data sets grow rapidly because they are increasingly gathered by cheap and numerous information-sensing Internet of Things (IoT) devices such as mobile devices, aerial (remote sensing), software logs, cameras, microphones, radio-frequency identification (RFID) readers and wireless sensor networks. Based on an IDC report prediction, the global data volume was predicted to grow exponentially from 4.4 zettabytes to 44 zettabytes between 2013 and 2020. By 2025, IDC predicts there will be 163 zettabytes of data. Further, with Covid 19, the last six month has seen an explosion of data and an acceleration in digital transformation mainly due to the scaling of remote work and particularly the rapid deployment and development of smart work from home technologies. In addition, there is a lot of online data circulating around smartphones, smartwatches, and IoT-enabled devices in our home and business.

[1] The byte is a unit of digital information that most commonly consists of eight bits.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.2.4.2 Data integration

Data integration revolves around combining data from different sources in order to provide users with a unified holistic view. Data integration is becoming more common as the volume of data (big data) and the necessity of sharing data increases. Generally, the data being integrated comes from a heterogeneous database system (information silos) and is transformed into a single coherent data store that provides synchronous data across a network of files for clients. One of the common uses of data integration can be seen in data mining when analysing and extracting information from existing databases that may be useful for Business information.

3.2.4.2.1 Description A data warehouse (DW or DWH) is a system used for reporting and data analysis, and it is considered a core component of business intelligence. DWs are central repositories of integrated data from one or more disparate sources, storing current and historical data in one single location;they are used to create analytical reports for workers throughout the enterprise. The data stored in the warehouse is uploaded from the operational systems and may pass through an operational data store. It may also require data cleansing for additional operations to ensure that the data is of the requisite quality before it is used in the DW for reporting. The two main approaches used to build a data warehouse system are Extract, Transform, Load (ETL) and Extract, Load, Transform (ELT). This is displayed in Figure (3.10).

Figure 3.10: Data warehouse.

However, the data warehouse approach is less feasible for data sets that are frequently updated as this would require the ETL process to be continuously re-executed for synchronisation. When implementing a warehouse, systematic differences or conflicts between tables to be integrated is called data heterogeneity. Data cleaning refers to the process of resolving such identification problems in the data. There are two types of data heterogeneity:

1. Structural heterogeneity: It occurs when the fields of the tuples in the database are structured differently in different databases. For example, in one database, the customer address might be recorded in one field named, say, addr, while, in another database, the same information might be stored in multiple fields such as street, city, state, and zipcode.

2. Lexical heterogeneity:It occurs when the tuples have identically structured fields across databases,but the data use different representations to refer to the same real-world object (such as StreetAddress = 44 W. 4th St. versus

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

StreetAddress = 44 West Fourth Street).

A data lake is a system or repository of data stored in its natural/raw format - usually object blobs or files. In general, it is a single store of data including raw copies of source system data, sensor data, social data, and transformed data used for tasks such as reporting, visualisation, advanced analytics and machine learning. A data lake may include structured data from relational databases (rows and columns), semi-structured data (CSV, logs, XML, JSON), unstructured data (emails, documents, PDFs) and binary data (images, audio, video). A data lake can be established on-premises (within an organisation's data centers) or in cloud services (using services from vendors such as Amazon, Microsoft, or Google).

A data hub (DH) is a collection of data from multiple sources organised for distribution, sharing, and often subsetting and sharing. DHs act as an integration point in a hub-and-spoke architecture. They are powered by an underlying multi-model database (which data lakes and virtual databases do not have), giving them the ability to serve as a system of truth with all the required enterprise security including data confidentiality (access control), data availability (HA/DR), and data integrity (distributed transactions) capabilities. They have the tools to curate the data such as enriching, mastering, and harmonising. They support operational and transactional applications. A data hub differs from a data warehouse in that it is generally unintegrated and often at different grains. It differs from an operational data store as it does not need to be limited to operational data. It also differs from a data lake by homogenising data and possibly serving data in multiple desired formats, rather than simply storing it in one place. Further it adds other value to the data such as de-duplication, quality, security, and a standardised set of query services. In general, it stores data in one place for availability, and allows/requires the consumer to process or add value to the data. A data hub is displayed in Figure (3.11).

Figure 3.11: Data hub.

Data Hubs are the ideal place for storing data within an enterprise. This is called enterprise data hub (EDH). They reduce the point-to-point connections made between callers and data suppliers and allow for the Data Hub organisation to negotiate deliverables and schedules with various data enclave teams. This is a more efficient environment than an organisational free-for-all in which different teams try to get new services and features from numerous other teams.

3.2.4.2.2 Smarter clouds Recent events have led to the scaling of remote work and particularly the rapid deployment and development of smart work from home technologies. This has allowed businesses to be accessed, delivered and enabled from anywhere. In addition, a lot of online data circulating around smartphones, smartwatches, and IoT-enabled devices in our homes and businesses. Processing and collecting all this data in one place will become

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

a complicated procedure. With the coming upsurge in data, organisations will need to enable better access to information, augment that information with better insight, and have the ability to respond quickly to the implications of that insight. Companies will need to address the challenges of exponential data growth, while also being proactive on issues like privacy, security and compliance.

Distributed computing technologies such as Kubernetes are making hybrid and distributed cloud infrastructure more manageable and operationally agile. Distributed cloud is where cloud services are distributed to different physical locations, but the operation, governance and evolution remain the responsibility of the public cloud provider. For instance, hybrid cloud architectures, which are a mixture of private cloud services, a public cloud, and/or on-premises infrastructure, imply a certain level of workload portability, orchestration, and management integration across these environments to provide a flexible infrastructure for an organisation's application. It gives businesses more data deployment options and greater flexibility by allowing the movement of workloads between public and private clouds as costs and computing needs change. This is displayed in Figure (3.12).

Figure 3.12: Hybrid cloud architecture.

With the explosion of IoT-enabled devices in our homes and businesses, processing and collecting all this data in one place is a challenge. It requires all information to be shared with cloud computing devices, and thus having no internet connection makes it much more complicated. One solution is to consider edge computing, which brings computation and data storage closer to the location where it is needed, to improve response times and save bandwidth. The aim being to move the computation away from the data centers and towards the edge of the network, exploiting smart objects, mobile phones or network gateways to perform tasks and provide services on behalf of the cloud. It offers servers and data storage for gadgets to access their devices and allow them to put data in them. Edge nodes help in creating a bridge between the cloud and the local service provider. This is displayed in Figure (3.13).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 3.13: Edge computing.

It is known as real-time data processing, and it is more efficient than cloud computing services. It will be used for interconnecting many smart devices and growing IoT technology.

3.3 Introduction to data mining

3.3.1 Data mining

Once the raw data retrieved from consumers has been integrated (see Section (3.2.4.2)), it has to be converted into coherent data for the business to study the market and perform economic analysis. This is called data mining. It is the process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems. It is an interdisciplinary subfield of computer science with an overall goal of extracting information (with intelligent method) from a data set and transforming that information into a comprehensible structure for further use (see Hastie et al. [2009], Kamber et al. [2011]). The rapid growth and integration of databases provided scientists, engineers, and business people with a vast new resource that can be analysed to make scientific discoveries, optimise industrial systems, and uncover financially valuable patterns. New methods targeted at large data mining problems have been developed. Hand et al. [2001] defined Data Mining (DM) as follows:

Definition 3.3.1 *Data Mining is the analysis of (large) observational data sets to find unsuspected relationships and to summarise the data in novel ways that are both understandable and useful to the data owner.*

More generally, data mining is the analysis step of the Knowledge Discovery in Databases process, or KDD (see Han et al. [2006]). The goal is the extraction of patterns and knowledge from large amounts of data, not the extraction (mining) of data itself. According to the Cross Industry Standard Process for Data Mining (CRISP-DM), the six phases of the knowledge discovery in databases (KDD) process are:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Business understanding
- Data understanding
- Data preparation
- Modelling
- Evaluation
- Deployment

or a simplified process such as

1. Pre-processing
2. Data Mining
3. Results Validation

Before data mining algorithms can be used, a target data set must be assembled. A common source for data is a data mart or data warehouse (see Section (3.2.4.2)). Pre-processing is essential to analyse the multivariate data sets before data mining (see Pyle [1999]). In presence of irrelevant and redundant information, or noisy and unreliable data, knowledge discovery during the training phase is more difficult. Data preparation and filtering steps can take considerable amount of processing time.

There exists several functionalities to data mining, such as:

1. Data characterisation summarising the general characteristics or features of a target class of data.
2. Data discrimination comparing the general features of target class data objects with the general features of objects from a set of contrasting classes.
3. Association analysis is the discovery of association rules showing attribute value conditions occurring frequently together in a given set of data.
4. Classification is the process of finding a set of models (or functions) that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown.

In fact, data mining involves six common classes of tasks (see Sondwale [2015]):

1. Anomaly detection (outlier/change/deviation detection): The identification of unusual data records, that might be interesting or data errors that require further investigation.
2. Association rule learning (dependency modelling): Searches for relationships between variables. For example, a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing purposes. This is sometimes referred to as market basket analysis.
3. Clustering: is the task of discovering groups and structures in the data that are in some way or another similar, without using known structures in the data.
4. Classification: is the task of generalising known structure to apply to new data. For example, an e-mail program might attempt to classify an e-mail as legitimate or as spam.
5. Regression: attempts to find a function which models the data with the least error that is, for estimating the relationships among data or datasets.
6. Summariation: providing a more compact representation of the data set, including visualisation and report generation.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.3.2 Data preparation

Machine learning (ML) (see Chapter (1)) requires the availability of coherent big data with computing infrastructure. Software platforms (see Section (1.7)) must store, analyse, and manage vast amounts of data in diverse datasets. Around $80 - 90\%$ of the time used for ML development is spent on data preparation or data pre-processing (see Stonebraker et al. [2019]). The end-to-end data flow from data collection to model serving is illustrated in Figure (3.14).

Figure 3.14: Data flow for machine learning.

Data collection and quality control, including the handling of biased and missing or dirty data, must be performed before model training. Data quality has a profound impact on model accuracy. We briefly introduce techniques for data cleaning and data validation for improving the data quality.

3.3.2.1 Data collection

Roh et al. [2019] presented a survey on data collection and identified three methods for data collection:

1. Data acquisition: It is the problem of finding the right datasets for training models. For instance, given the increasing number of datasets, searching for the right one becomes a challenge. There are several approaches to data acquisition:
   - Data discovery: It is the problem of indexing and searching datasets that may exist in corporate data lakes or the Web.
   - Data augmentation: It focuses on complementing existing datasets by integrating them with external data.
   - Data generation: In case of missing data, one can create datasets using crowd-sourcing or synthetic data generation techniques.
2. Data labelling: It is necessary for supervised and semi-supervised learning applications. Various scalable techniques have been proposed using
   - semi-supervised learning: It uses existing labels to predict the other labels as much as possible.
   - crowd-sourcing: Workers manually label examples or more advanced techniques such as active learning where the manual labelling is done only for examples that are most likely to benefit the model's accuracy.
   - weak supervision: The idea is to semi-automatically generate labels that are not perfect (weak labels), but at scale where the larger volume may compensate for the lower label quality. It is useful in applications with few or no labels to start with.
3. Improving existing data and models: One can improve the quality of existing data or use transfer learning to re-use existing models rather than training from scratch. In the latter, we make model training be more robust to dirty data. Further, one can also re-use existing models using transfer learning techniques.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

This is illustrated with the technique of leaf nodes in Figure (3.15).

Figure 3.15: Data collection for machine learning.

3.3.2.2 Data pre-processing

Training data contains various errors that need to be detected and fixed. For instance, the data maybe dirty, the data may change as the data source itself evolves, the data maybe missing due to possible bugs in the data source. Data preprocessing includes cleaning, instance selection, normalisation, transformation, feature extraction and selection. Data validation focuses on finding problems in the data affecting the machine learning pipeline. We must also consider data sanitisation against poisoning, and improving model fairness.

Data cleansing, or data cleaning, is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data. Data cleansing may be performed interactively with data wrangling tools, or as batch processing through scripting. After cleansing, a data set should be consistent with other similar data sets in the system.

- The actual process of data cleansing may involve removing typographical errors or validating and correcting values against a known list of entities.
- Some data cleansing solutions will clean data by cross checking with a validated data set. Another approach is data enhancement, where data is made more complete by adding related information. For example, appending addresses with any phone numbers related to that address.
- Data cleansing may also involve activities like, harmonisation of data, and standardisation of data. For example, harmonisation of short codes (st, rd, etc.) to actual words (street, road, etcetera). Standardisation of data is a means of changing a reference data set to a new standard, ex, use of standard codes.

However, focusing only on fixing the data does not guarantee the best model accuracy. The cleaned data must satisfy some quality criteria. High-quality data needs to pass a set of quality criteria such as:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Validity: The degree to which the measures conform to defined business rules or constraints.
- Accuracy: The degree of conformity of a measure to a standard or a true value. In general, it is very hard to achieve through data-cleansing, as it requires accessing an external source of data that contains the true value: such "gold standard" data is often unavailable.
- Completeness: The degree to which all required measures are known. Incompleteness is almost impossible to fix with data cleansing methodology. One can not infer facts that were not captured when the data in question was initially recorded.
- Consistency: The degree to which a set of measures are equivalent in across systems. Inconsistency occurs when two data items in the data set contradict each other. For instance, a customer is recorded in two different systems as having two different current addresses, and only one of them can be correct. Fixing inconsistency is not always possible as it requires a variety of strategies. For example, deciding which data were recorded more recently, which data source is likely to be most reliable (the latter knowledge may be specific to a given organisation), or simply trying to find the truth by testing both data items.
- Uniformity: The degree to which a set data measures are specified using the same units of measure in all systems. For example, in datasets pooled from different locales, weight may be recorded either in pounds or kilos, and must be converted to a single measure using an arithmetic transformation.

Contrary to dirty data, data poisoning is a malicious intention of making the model fail. While early work focused on specific applications such as spam detection and sensors, recent studies tend to focus on specific models. Cretu et al. [2008] used training metadata as data sanitisation method in the case of attacks occurring in relatively confined time intervals. More recently, adversarial machine learning, which attempts to fool models through malicious inputs, have become popular.

3.3.3 Knowledge discovery

3.3.3.1 Pattern recognition

In the more general problem of pattern recognition, we distinguish

- Classification and clustering, which is the assignment of some sort of output value to a given input value
- Regression, which assigns a real-valued output to each input
- Sequence labelling, which assigns a class to each member of a sequence of values (for example, part of speech tagging, which assigns a part of speech to each word in an input sentence)
- Parsing, which assigns a parse tree to an input sentence, describing the syntactic structure of the sentence

Often, the individual observations are analysed into a set of quantifiable properties, known variously as explanatory variables or features. These properties may variously be

- categorical (such as A, B, AB or O, for blood type),
- ordinal (such as large, medium or small),
- integer-valued (such as the number of occurrences of a particular word in an email), or
- real-valued (such as a measurement of blood pressure).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Terminology across fields is quite varied. In statistics, where classification is often done with logistic regression or a similar procedure, the properties of observations are termed explanatory variables (or independent variables, regressors, etc.), and the categories to be predicted are known as outcomes, which are considered to be possible values of the dependent variable. In machine learning, the observations are often known as instances, the explanatory variables are termed features (grouped into a feature vector), and the possible categories to be predicted are classes.
Class label is the discrete attribute having finite values (dependent variable) whose value you want to predict based on the values of other attributes (features). The class label always takes on a finite number of different values. Classification is a type of problem whereas labelling is a function trying to label an object and classify using the information. For instance, given a set of examples of the form $(\text{attribute values}\ ,\ \text{class label})$, we want to learn a rule that computes the label from the attribute values. For example, in binary classification, we use positive $(+1)$ and negative (-1) to denote the two class labels.

3.3.3.2 Predictive modelling

In general, a model (see Section (1.1)) is a predictive model that we want to construct or discover from the data set. A data mining model gets data from a mining structure and then analyses that data by using a data mining algorithm. A mining model stores information derived from statistical processing of the data, such as the patterns found as a result of analysis. After a mining model has been processed, it contains meta data, results, and bindings back to the mining structure.
One major task of pattern recognition and data mining, within machine learning, is to construct good models from data sets (see Otero et al. [2013]). We can use the same data to create multiple models, using perhaps a clustering algorithm, decision tree algorithm (classification), and naive Bayes algorithm. Each model type creates different set of patterns, item sets, rules, or formulas, which we can use for making predictions. Generally each algorithm analyses the data in a different way, so the content of the resulting model is also organised in different structures. In one type of model, the data and patterns might be grouped in clusters. In another type of model, data might be organised into trees, branches, and the rules that divide and define them.

The process of generating models from data is called learning or training. There are several types of learning, such as

- Supervised learning: it consists of a specified set of classes, and example objects labelled with the appropriate class. The goal being to learn from the training objects, enabling novel objects to be identified as belonging to one of the classes.
- Unsupervised learning: it is the task of inferring a function that describes the structure of unlabelled data (data that has not been classified or categorised). Among neural network models, the self-organizing map (SOM) and adaptive resonance theory (ART) are commonly used in unsupervised learning algorithms.
- Deep reinforcement learning: deep models allow reinforcement learning algorithms to solve complex problems end to end.
- Limited supervision: we know what we want, but do not know how to get it.
- Sequential decision making: take advantages of advances in deep learning, reinforcement learning and computational capability.

The learned model is called a predictor or a hypothesis (or learner). There are several types of label such as categorical or numerical.

1. If the label is categorical, such as shape, the task is also called classification and the learner is also called classifier.
2. However, if the label is numerical, such as x-coordinate, the task is also called regression and the learner is also called fitted regression model.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The predictive models are the supervised learning functions. Some of these functions with associated algorithms are:

- classification: naive Bayes, adaptive Bayes network, support vector machine.
- regression: support vector machine.
- attribute importance: minimal descriptor length.

3.3.3.3 Model training

Model learning (or training) is the process of generating models from data. There are different types of model training such as

- Fair training: It addresses data bias which results in discriminative mode behaviour. There are several definitions of fairness that can be categorised as group or individual measures. The former compare different sensitive groups and make sure their statistics are similar. The latter ensure that a prediction of an individual does not deviate too much from that of a similar person. One solution is to mitigate the unfairness (1) before model training on the data (pre-processing), (2) during model training (in-processing), (3) after model training (post-processing). In general there is a trade-off between accuracy and fairness when the two objectives differ.
- Robust training: It copes with dirty data (after data cleaning) and still produces accurate results. Several deep learning techniques developed to consider the existence of data noises and errors. Some examples include:
  - Noisy or missing labels: some techniques are (1) loss correction estimates the confidence of a label for each sample and adjusts the loss for the sample based on its label confidence during backward propagation, (2) sample selection estimates the confidence of a label for each sample and includes the samples in training only if their label confidence is above some threshold. Regarding missing labels, semi-supervised learning builds a model from a mixture of labelled and unlabelled data, by adopting unsupervised loss or collaborating with mix-up augmentation for unlabelled data.
  - Missing data: It can reduce the statistical power of a model and produce biased estimates. One solution is to consider recurrent neural networks (RNNs), which are used for time-series data. They have been extended to receive the context about missing values. For instance, Che et al. [2018] proposed GRU-D to analyse medical data.
- Representation learning: It transforms the data in the best embedding format for model training. Data embedding transform or convert input data to improve its learning suitability. The best embedding is obtained in an end-to-end training procedure. Data embedding can be applied in natural language processing and on spatial data. For instance, Park et al. [2020] used a coordinate in a high dimensional embedding space rather than the original latitude and longitude coordinate to feed a location to a DNN model. They achieved high enough dimensionality to have the descriptive capacity in the DNN model.

3.3.4 Clustering

3.3.4.1 Introduction

3.3.4.1.1 Overview Cluster analysis, or clustering, is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). It was introduced to psychology by Zubin in 1938 and Tryon [1939]. Cattell [1943] used it for trait theory classification in personality psychology. Clustering algorithms are used to classify unlabelled data points into groups based on similar features (see Everitt [2011]). Clustering can be achieved by various algorithms that differ significantly in their understanding of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances between cluster members, dense areas of the data space, intervals or particular

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

statistical distributions. Clustering can therefore be formulated as a multi-objective optimisation problems (see Section (19.4)). Cluster analysis is not an automatic task, but an iterative process of knowledge discovery or interactive multi-objective optimisation involving trial and failure. It is often necessary to modify data preprocessing and model parameters until the result achieves the desired properties. For instance, conventional text clustering pipelines involves preprocessing raw text strings, followed by feature extraction, and then applying a clustering algorithm (see Kaleel et al. [2015]). Recent advances in machine learning enable efficient representation learning from raw data in a hierarchical manner (see Bengio et al. [2013]). Recently, several NLP tasks benefited from deep-neural approaches, including sentiment classification of tweets (see Severyn et al. [2015]), simulating epidemics from Twitter. Gencoglu [2018] proposed deep convolutional autoencoders (CAEs) for obtaining efficient representations of health related tweets in an unsupervised manner.

3.3.4.1.2 Some algorithms Some popular algorithms are:

- Centroid based algorithms: K-means, Gaussian mixture models (GMM), Fuzzy c-mean and any other variation that uses Expectation-Maximisation (EM) for finding parameters (see Jordan et al. [1995]). Knowledge of a number of clusters is essential here. Most of them are $O(N)$, but initialisation is a problem in them.
- Connectivity based algorithms: All the hierarchical clustering algorithms, most of them are $O(N^2)$ or more, but one can stop at any arbitrary number of clusters. Hierarchy of clusters is represented as a tree, the algorithm is not sensitive to the choice of distance metric, suitable when the underlying data has a hierarchical structure.
- Density based algorithms: These methods consider the clusters as the dense region having some similarity and different from the lower dense region of the space. They include Density-based spatial clustering of application with noise (DBSCAN), ordering points to identify clustering structure (OPTICS). Discover the clusters and identify outliers as noises, perform poorly when the groups are of varying density. This drawback also occurs with very high-dimensional data.
- Probabilistic: Latent Dirichlet Allocation (LDA). Topic modelling is a type of statistical modelling for discovering the abstract topics that occur in a collection of documents. LDA is an example of a topic model used to classify text in a document to a particular topic. It builds a topic per document model, and words per topic model, modelled as Dirichlet distributions.
- Grid-based methods: In this method the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operation done on these grids are fast and independent of the number of data objects example STING (Statistical Information Grid), wave cluster, CLIQUE (Clustering in Quest).
- Dimensionality Reduction: tSNE, PCA, kernel PCA (KPCA) followed by visualisation.
- Neural Networks/Deep Learning: Autoencoders, Self-organising Maps (SOM)

The most appropriate clustering algorithm for a particular problem often needs to be chosen experimentally, unless there is a mathematical reason to prefer one cluster model over another. An algorithm that is designed for one kind of model will generally fail on a data set that contains a radically different kind of model. For example, K-means cannot find non-convex clusters.

3.3.4.2 Probabilistic models

When mining text corpora, or other collections of discrete data, we want to infer short descriptions of the members of a collection in order to efficiently process large collections while preserving the essential statistical relationships useful for basic tasks such as classification, summarisation, or similarity and relevance judgements. In information retrieval (IR), one method is to reduce each document in the corpus to a vector of real numbers, where each one represents ratios of counts (see Salton et al. [1983]). For instance, in the tf-idf scheme, a basic vocabulary or words (or terms)

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

is chosen, and for each document, a count is formed of the number of occurrences of each word. After normalisation, this term frequency count is compared to an inverse document frequency count measuring the number of occurrences of a word in the entire corpus. Eventually, we get a term-by-document matrix X whose columns contain the tf-idf values for each of the documents in the corpus. Hence, the tf-idf scheme reduces documents of arbitrary length to fixed-length lists of numbers. However, this approach provides a relatively small amount of reduction in description length, and does not reveal much about the way of inter or intra document statistical structure. IR proposed several other dimensionality reduction techniques such as latent semantic indexing (LSI) (see Deerwester et al. [1990]). It uses a singular value decomposition of the matrix X to identify a linear subspace in the space of tf-idf features that captures most of the variance in the collection. The authors argued that the derived features of LSI, which are linear combinations of the original tf-idf features, could capture some aspects of basic linguistic notions such as synonym and polysemy. Papadimitriou et al. [1998] developed a generative probabilistic model of text corpora and studied the ability of LSI to recover some aspects of the generative model from data.

An alternative approach is to fit the model to data using maximum likelihood or Bayesian methods (see Gelman et al. [1995]). Hofmann [1999] proposed the probabilistic LSI (pLSI) which models each word in a document as a sample from a mixture model (see Section (4.3.3.2)), where the components are multinomial random variables that can be viewed as representations of topics. Thus, each word is generated from a single topic, and different words in a document may be generated from different topics. Each document is represented as a list of mixing proportions (weights) for these mixture components and thereby reduced to a probability distribution on a fixed set of topics. This distribution is the reduced description associated with the document. However, this approach does not provide a probabilistic model at the level of documents. The model was extended by Blei et al. [2003] by introducing a Dirichlet prior on the mixture weights θ, which was called Latent Dirichlet Allocation (LDA). See details in Section (11.5).

The main assumptions behind the LSI and pLSI are based on the bag-of-words (BoWs) assumption that the order of words in a document can be neglected, which is an assumption of exchangeability for the words in a document. Further, it is also assumed that documents are exchangeable. From de Finetti's theorem, any collection of exchangeable random variables has a representation as a mixture distribution, in general an infinite mixture (see Accardi [1994]). Thus, to obtain an exchangeable representation for documents and words, one must consider mixture models that capture the exchangeability of both words and documents. This is the idea behind the latent Dirichlet allocation (LDA) model. Here, exchangeability means conditionally independent and identically distributed, with condition on a latent parameter of a probability distribution. Conditionally, the joint distribution of random variables is simple and factored over the latent parameter. Bur the joint distribution can be complex. This is different from the assumption that the random variables are independent and identically distributed (i.i.d.).

3.3.4.3 Features representation

In NLP, when working with text corpora, one has to pre-process raw text strings and perform features extraction. Then, one applies a clustering algorithm. In general, the techniques used for representing text as numerical vectors are bag-of-words (BoWs) and term frequency inverse document frequency (tf-idf). These methods are based on word occurrence counts and may result in sparse document-term matrix (see Salton et al. [1983]). Some algorithms proposed for clustering documents into topics include variants of hierarchical, density-based, and centroid-based clustering methods. The K-means algorithm is quite popular. When it comes to topic modelling, one generally consider generative models, which assume that a document is a mixture of topics, where a topic is a probability distribution over words (see Section (3.3.4.2)). While topic modelling is slightly different from clustering, representing a tweet as a topic vector is a way of reducing dimensionality, or feature extraction, which can be followed by a clustering algorithm.

Alternatively, one can learn representations from raw data by using neural networks. For instance, it has been used in domains such as computer vision and NLP (see Hinton et al. [2006], Bengio et al. [2013]). It relies on the hierarchical,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

layer-wise architecture of neural networks in which the raw input data is encoded into informative representations of lower dimensions in a highly non-linear fashion. Examples of deep neural network architectures specifically used for representation learning are autoencoders, denoising autoencoders, convolutional autoencoders, sparse autoencoders, stacked autoencoders and other combinations such as denoising convolutional autoencoders. In an autoencoder training, the network tries to reconstruct the input data at its output, forcing the model to capture the most salient features of the data at its intermediate layers. If the latter correspond to a lower dimensional latent space than the original input, the autoencoders are known as undercomplete. Activations extracted from these layers can be regarded as compact, non-linear representation of the input.

Another significant advancement in NN representation learning in NLP tasks is word embeddings (or distributed representation of words). By representing each word in a given vocabulary with a real-valued vector of fixed dimension, word embeddings enable capturing of lexical, semantic, or even syntactic similarities between words. These vector representations are learned from large corpora and can be used to enhance the performance of numerous NLP tasks such as document classification, question answering and machine translation. Some frequently used word embeddings are word2vec (see Mikolov et al. [2013]) and GloVe (Global Vectors for Word Representation) (see Pennington et al. [2014]). In both cases, the vectors are extracted in an unsupervised manner, and are based on the distributional hypothesis, where words occurring in the same contexts tend to have similar meanings. Both methods have been used for topical clustering of tweets (see Dai et al. [2017]) and topic modelling (see Nguyen et al. [2018]).
Bojanowski et al. [2016] presented fastText, which treats each word as a bag of character n-grams, producing better results for rare words. New advances and enhanced methods for deducting better word and sentence representations were recently proposed. Peters et al. [2018] introduced the ELMo, and Devlin et al. [2018] proposed the BERT, which we detail in Section (14.3.1).

3.3.5 Classification

In this section, we will focus on supervised learning, especially classification.

3.3.5.1 Definitions

In machine learning and statistics, classification is the problem of identifying to which of a set of categories (subpopulations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known (see Alpaydin [2010]). Classification is considered an instance of supervised learning, that is, learning where a training set of correctly identified observations is available. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of inherent similarity or distance.
Machine learning have been traditionally used for classification. A classification process usually consists of three main stages:

1. In the first stage, data from objects have to be collected for the design of classifiers.
2. In the second stage, feature extraction is performed to extract characteristics from the collected data to be classified such that redundant information is removed and representative information is extracted resulting in reduction of input dimensions and improved classification accuracy.
3. In the third stage, a classifier is designed using the feature data.

Classification can be thought of as two separate problems:

1. binary classification: only two classes are involved (it is a better understood task).
2. multiclass classification: involves assigning an object to one of several classes. Since many classification methods have been developed specifically for binary classification, multiclass classification often requires the combined use of multiple binary classifiers.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.3.5.2 Some examples

3.3.5.2.1 Anomaly detection An outlier is identified as any data object or point that significantly deviates from the remaining data points. Anomaly detection (or outlier detection) is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data. Typically, the anomalous items will translate to some kind of problem such as bank fraud, a structural defect, medical problems or errors in a text. Anomalies are also referred to as outliers, novelties, noise, deviations and exceptions.
The three main categories of anomaly detection techniques are:

1. Supervised anomaly detection techniques require a data set labelled as "normal" and "abnormal" and involves training a classifier. The key difference to many other statistical classification problems is the inherent unbalanced nature of outlier detection.

2. Semi-supervised anomaly detection techniques construct a model representing normal behaviour from a given normal training data set, and then test the likelihood of a test instance to be generated by the learnt model.

3. Unsupervised anomaly tetection techniques detect anomalies in an unlabelled test data set under the assumption that the majority of the instances in the data set are normal by looking for instances that seem to fit least to the remainder of the data set.

The three main types of anomaly are

1. Point anomalies: They are individual instances that are anomalous with respect to the majority of other individual instances.

2. Conditional anomalies (a.k.a. contextual anomalies), also refer to individual anomalous instances, but in a specific context, that is, data instances are anomalous in the specific context, otherwise they are normal.

3. Group anomalies (a.k.a. collective anomalies), are subset of data instances anomalous as a whole with respect to the other data instances.

There are several challenges that make anomaly detection difficult:

- The amount of data for training and testing the model may be limited, ans it may not be labelled (we do not know which data points are anomalies). ML algorithms typically require large amounts of data. Since anomalies are by definition statistically unlikely, datasets are often imbalanced (there are more occurrences of normal behaviour than of anomalous behaviour), which presents additional challenges in training models that accurately identify or predict anomalies.

- Anomaly detectors may be built on dynamic systems with rapidly growing user bases. As a result, anomaly detectors have to adapt their behaviour over time, as the underlying system evolves.

- Is the data showing stationary behaviour (their mean and variance do not change over time), or does it often change over years and seasons, exhibiting trends and seasonality.

- Use ML for multivariate anomalies.

Deep anomaly detection aims at learning feature representations or anomaly scores via neural networks. The three main frameworks are

1. Deep learning for generic feature extraction (dimensionality reduction): Extract low-dimensional feature representations from high-dimensional and/or non-linearly separable data for downstream anomaly detection. Autoencoder (AE) is one type of neural network that projects its input to a lower-dimensional space and then reconstructs back the data in the output.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. Learning representations of normality: Couple feature learning with anomaly scoring in some ways. There are two main groups: (1) generic feature learning, (2) anomaly measure-dependent feature learning.

3. End-to-end anomaly score learning: This category of methods aims at learning an end-to-end anomaly score learning network.

Deep learning has superior capability in capturing complex temporal-spatial dependence and learning representations of a set of unordered data points. Most shallow and deep models for anomaly detection assume that the abnormality of data instances is independent and identically distributed (iid), while the abnormality in real applications may suffer from some non-iid characteristics (such as multifractal). On majority, the problems and tasks are regular with evident patterns. However, for a minority of them, events are uncertain and rare, leading to some unique complexities that render general deep learning techniques ineffective.

3.3.5.3 The algorithms

Most algorithms describe an individual instance whose category is to be predicted using a feature vector of individual, measurable properties of the instance. Some algorithms work only in terms of discrete data and require that real-valued or integer-valued data be discretised into groups (e.g. less than 5, between 5 and 10, or greater than 10). Other classifiers work by comparing observations to previous observations by means of a similarity or distance function.
In the literature, classification techniques and methods from traditional methods to machine learning methods can be found. Examples of the former are Linear discriminant analysis (LDA), logic based method, statistical approach, or instance-based methods. In the latter, models such as decision trees, SVM, neural networks, Bayesian belief networks, genetic algorithm etc have been considered. The ability to perform classification and be able to learn to classify objects is paramount to the process of decision making.
The process of seeking relationships within a data set involves a number of steps:

1. Model or pattern structure: determining the nature and structure of the representation to be used.

   (a) a model structure is a global summary of a data set making statements about any point in the full measurement space,

   (b) pattern structures only make statements about restricted regions of the space spanned by the variables.

2. Score function: deciding how to quantify and compare how well different representations fit the data (choosing a score function).
   Score functions judge the quality of a fitted model, and should precisely reflect the utility of a particular predictive model.

3. Optimisation and search method: choosing an algorithmic process optimising the score function.

   (a) optimisation problem: the task of finding the best values of parameters in models

   (b) combinatorial problem: the task of finding interesting patterns (such as rules) from a large family of potential patterns, and is often accomplished using heuristic search techniques.

4. Data management strategy: deciding what principles of data management are required for implementing the algorithms efficiently.
   Data management strategy is about the ways in which the data are stored, indexed, and accessed.

Some examples of classification algorithms include:

- Linear classifiers: Fisher's linear discriminant (see Fisher [1936]), Logistic regression, Naive Bayes classifier (see Binder [1978]), Perceptron

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Support vector machines: Least squares support vector machines
- Quadratic classifiers
- Kernel estimation: k-nearest neighbour
- Boosting (meta-algorithm)
- Decision trees: Random forests
- Neural networks
- Learning vector quantization

The success of classification learning is heavily dependent on the quality of the data provided for training, as the learner only has the input to learn from. On the other hand, we want to avoid overfitting the given data set, and would rather find models or patterns generalising potential future data. Note, even though data mining is an interdisciplinary exercise, it is a process relying heavily on statistical models and methodologies. The main difference being the large size of the data sets to manipulate, requiring sophisticated search and examination methods. Further difficulties arise when there are many variables (curse of dimensionality), and often the data is constantly evolving. Recently, a lot of advances have been made on machine learning strategies mimicking human learning, and we refer the reader to Battula et al. [2013] for more details.

3.3.6 The challenges of computational learning

While the ability to perform classification and be able to learn to classify objects is paramount to the process of decision making, there exists several types of learning (see Mitchell [1997]), such as

- Supervised learning: learning a function from example data made of pairs of input and correct output.
- Unsupervised learning: learning from patterns without corresponding output values
- Reinforcement learning: learning with no knowledge of an exact output for a given input. Nonetheless, online or delayed feedback on the desirability of the types of behaviour can be used to help adaptation of the learning process.
- Active learning: learning through queries and responses.

More formally, these types of learning are part of what is called inductive learning where conclusions are made from specific instances to more general statements. That is, examples are provided in the form of input-output pairs $[X, f(X)]$ and the learning process consists of finding a function h (called hypothesis) which approximates a set of samples generated by the function f. The search for the function h is formulated in such a way that it can be solved by using search and optimisation algorithms.
Some of the challenges of computational learning can be summarised as follow:

- Identifying a suitable hypothesis can be computationally difficult.
- Since the function f is unknown, it is not easy to tell if the hypothesis h generated by the learning algorithm is a good approximation.
- The choice of a hypothesis space describing the set of hypotheses under consideration is not trivial.

As a result, a simple hypothesis consistent with all observations is more likely to be correct than a complex one. In the case where multiple hypotheses (an Ensemble) are generated, it is possible to combine their predictions with the aim of reducing generalisation error. For instance, boosting works as follow

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Examples in the training set are associated with different weights.
- The weights of incorrectly classified examples are increased, and the learning algorithm generates a new hypothesis from this new weighted training set. The process is repeated with an associated stopping criterion.
- The final hypothesis is a weighted-majority of all the generated hypotheses which can be based on different mixture of expert rules.

The difficult part being to know when to stop the iterative process and how to define a proper measure of error. One way forward is to consider the Probably Approximately Correct (PAC) learning which can be described as follow:

- A hypothesis is called approximately correct if its error insample lies within a small constant of the true error.
- By learning from a sufficient number of examples, one can calculate if a hypothesis has a high probability of being approximately correct.
- There is a connection between the past (seen) and the future (unseen) via an assumption stating that the training and test datasets come from the same probability distribution. It follows from the common sense that non-representative samples do not help learning.

More formally, a concept class C is said to be PAC learnable using a hypothesis class H if there exists a learning algorithm L such that for all concepts in C, for all instance distributions D on an instance space X,

$$\forall \epsilon \text{ , } \delta \text{ such that } 0 < \epsilon \text{ , } \delta < 1$$

when given access to the example set, produces with probability at least $(1-\delta)$, a hypothesis h from H with error no-more than ϵ. To specify the problem we get a set of instances X, a set of hypotheses H, a set of possible target concepts C, training instances generated by a fixed, unknown probability distribution $\mathcal{D}$ over X, a target value $c(x)$, some training examples $< x, c(x) >$, and a hypothesis h estimating c. Then, the error of a hypothesis h satisfies

$$error_D = P_{x \in \mathcal{D}}(c(x) \neq h(x))$$

and the deviation of the true error from the training error satisfies

- Training error: $h(x) \neq c(x)$ over training instances.
- True error: $h(x) \neq c(x)$ over future random instances.

We must now measure the difference between the true error and the training error. Any hypothesis h is consistent when for all training samples,

$$h(x) = c(x)$$

If the hypothesis space H is finite, and $\mathcal{D}$ is a sequence of $m \geqslant 1$ independent random examples of some target concept c, then for any $0 \leqslant \epsilon \leqslant 1$, the probability that $VS_{H,D}$ contains a hypothesis with error greater than ϵ is less than

$$|H|e^{-\epsilon m}$$

and $P(1 \text{ of } |H| \text{ hyps. consistent with m exs.}) < |H|e^{-\epsilon m}$. Considering a bounded sample size, for the probability to be at most δ, that is, $|H|e^{-\epsilon m} \leqslant \delta$, then

$$m \geqslant \frac{1}{\epsilon}\big(\ln|H| + \ln\frac{1}{\delta}\big)$$

More can be found on agnostic learning and infinite hypothesis space in books by Mitchell [1997] and Bishop [2006].

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.3.7 Evaluation and validation

3.3.7.1 Overview

Data mining can unintentionally be misused, and can then produce results which appear to be significant, but which do not actually predict future behaviour and cannot be reproduced on a new sample of data and bear little use. Often this results from investigating too many hypotheses and not performing proper statistical hypothesis testing. Not all patterns found by the data mining algorithms are necessarily valid. It is common for the data mining algorithms to find patterns in the training set which are not present in the general data set. This is called overfitting. To overcome this problem, the evaluation uses a test set of data on which the data mining algorithm was not trained (see Section (1.6.2)). The learned patterns are applied to this test set, and the resulting output is compared to the desired output.

Definition 3.3.2 *Model validation is referred to as the process where a trained ML model is evaluated with a testing data set. The latter is a separate portion of the same data set from which the training set is derived.*

A number of statistical methods may be used to evaluate the algorithm, such as Precision and Recall which are evaluated from True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). They are defined as follows:

- Precision (also called positive predictive value): It is the fraction of relevant instances among the retrieved instances.
- Recall (also known as sensitivity): It is the fraction of relevant instances that were retrieved.

Both precision and recall are therefore based on relevance. This is illustrated in Figure (3.16).

Figure 3.16: Precision and recall.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

More recently, Sensitivity, Specificity, Accuracy and Receiver Operating Characteristic (ROC) curves have been used to evaluate the tradeoff between true- and false-positive rates of classification algorithms.
Sensitivity and Specificity are statistical measures of the performance of a binary classification test, also known in statistics as a classification function:

- Sensitivity (also called the true positive rate, the recall, or probability of detection in some fields) measures the proportion of actual positives that are correctly identified as such (e.g., the percentage of sick people who are correctly identified as having the condition).
- Specificity (also called the true negative rate) measures the proportion of actual negatives that are correctly identified as such (e.g., the percentage of healthy people who are correctly identified as not having the condition).

In the terminology true/false positive/negative, true or false refers to the assigned classification being correct or incorrect, while positive or negative refers to assignment to the positive or the negative category. That is,

- True positives (TP): The correctly predicted positive values.
- True negatives (TN): The correctly predicted negative values.
- False positives (FP): The incorrectly predicted positive values.
- False negatives (FN): The incorrectly predicted negative values.

For instance, in medicine sensitivity, specificity and accuracy are widely used statistics to describe a diagnostic test since they quantify how good and reliable a test is.

- Sensitivity evaluates how good the test is at detecting a positive disease
- Specificity estimates how likely patients without disease can be correctly ruled out
- The ROC curve is a graphic presentation of the relationship between both sensitivity and specificity helping to decide the optimal model through determining the best threshold for the diagnostic test
- Accuracy measures how correct a diagnostic test identifies and excludes a given condition. Accuracy of a diagnostic test can be determined from sensitivity and specificity with the presence of prevalence.

The terms used are true positive (TP), true negative (TN), false negative (FN), and false positive (FP). If a disease is proven present in a patient, the given diagnostic test also indicates the presence of disease, the result of the diagnostic test is considered true positive. Similarly, if a disease is proven absent in a patient, the diagnostic test suggests the disease is absent as well, the test result is true negative (TN). In machine learning, the problem of statistical classification is summarised in a confusion matrix (or accuracy table), which is a specific table layout allowing for visualisation of the performance of the algorithm. Each row of the table represents the instances in a predicted class, while each column represents the instances in an actual class (or vice versa). It is a contingency table with two dimensions: actual and predicted. See details in Table (3.2).
Both true positive and true negative suggest a consistent result between the diagnostic test and the proven condition (also called standard of truth). If the diagnostic test indicates the presence of disease in a patient who actually has no such disease, the test result is false positive (FP). Similarly, if the result of the diagnosis test suggests that the disease is absent for a patient with disease for sure, the test result is false negative (FN). Both false positive and false negative indicate that the test results are opposite to the actual condition.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Outcome of the test	Condition as determined by the Standard of Truth		
	Positive	Negative	Row total
Positive	TP	FP	$TP + FP$
Negative	FN	TN	$FN + TN$
Column total	$TP + FN$	$FP + TN$	$N = TP + TN + FP + FN$

Table 3.2: Accuracy table

3.3.7.2 Measures of performance

The evaluation measures are described in terms of TP, TN, FN and FP. The Accuracy is the ratio of correctly predicted observations to the total observations. It is given by

$$\text{Accuracy} = \frac{TN + TP}{TN + TP + FN + FP}$$

It is a useful measure only when we have symmetric datasets, where values of false positives and false negatives are almost identical. Otherwise, we need to consider other metrics to measure the performance of the algorithm (see Powers [2011]). The Precision and Recall are defined as follows:

- Precision: It is the ratio of correctly predicted positive (negative) observations to the total predicted positive (negative) observations. High precision relates to the low positive rate.
- Recall: It is the ratio of correctly predicted positive (negative) observations to all observations in the actual class.

They are given by:

$$\begin{aligned}
\text{Precision}_{+} &= \frac{TP}{TP + FP} \\
\text{Precision}_{-} &= \frac{TN}{TN + FN} \\
\text{Recall}_{+} &= \frac{TP}{TP + FN} \\
\text{Recall}_{-} &= \frac{TN}{TN + FP}
\end{aligned}$$

Precision is the weighted average of precision positive and negative, while Recall is the weighted average of recall positive and negative.
$F1$-Score is the weighted average of Precision and Recall, taking both false positives and false negatives into account. It is more useful than Accuracy in cases where we have uneven class distribution. The F-measure is given by

$$\text{F-measure} = \frac{2 \times \text{ Precision } \times \text{ Recall}}{\text{Precision } + \text{ Recall}}$$

The value range of these measures is between 0 and 1, where 0 indicates the worst performance, while 1 indicates the best performance.
Sensitivity and Specificity are defined as follows:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}\text{Sensitivity} &= \frac{TP}{TP+FN} = \frac{TP}{P}\\ \text{Specificity} &= \frac{TN}{TN+FP} = \frac{TN}{N}\end{aligned}$$

where $P = TP + FN$ is the condition positive and $N = TN + FP$ is the condition negative. Thus, sensitivity is the proportion of true positives that are correctly identified by a diagnostic test, showing how good the test is at detecting a disease. The numerical values of sensitivity represents the probability of a diagnostic test identifies patients who do in fact have the disease. The higher the numerical value of sensitivity, the less likely diagnostic test returns false-positive results. Specificity is the proportion of the true negatives correctly identified by a diagnostic test, suggesting how good the test is at identifying normal (negative) condition. The numerical value of specificity represents the probability of a test diagnoses a particular disease without giving false-positive results. Accuracy is the proportion of true results, either true positive or true negative, in a population. It measures the degree of veracity of a diagnostic test on a condition.
Note, accuracy can be determined from sensitivity and specificity, where prevalence is known. Prevalence is the probability of disease in the population at a given time

$$\text{Accuracy} = \text{Sensitivity} \times \text{Prevalence} + \text{Specificity} \times \big(1 - \text{Prevalence}\big)$$

Note, even if both sensitivity and specificity are high, say 99%, it does not suggest that the accuracy of the test is equally high as well.

A receiver operating characteristic curve (ROC curve), is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied (see Fawcett [2006]). The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. In machine learning, the true-positive is also known as Sensitivity, Recall or probability of detection. The false-positive rate is also known as probability of false alarm and can be calculated as $(1 - \text{ Specificity })$. It can also be thought of as a plot of the power as a function of the Type I Error of the decision rule (when the performance is calculated from just a sample of the population, it can be thought of as estimators of these quantities). The ROC curve is thus the sensitivity or recall as a function of fall-out. In general, if the probability distributions for both detection and false alarm are known, the ROC curve can be generated by plotting the cumulative distribution function (area under the probability distribution from $-\infty$ to the discrimination threshold) of the detection probability in the y-axis versus the cumulative distribution function of the false-alarm probability on the x-axis. ROC analysis provides tools to select possibly optimal models and to discard suboptimal ones independently from (and prior to specifying) the cost context or the class distribution. ROC analysis is related in a direct and natural way to cost/benefit analysis of diagnostic decision making.
We let $P = TP + FN$ be the condition positive and $N = TN + FP$ be the condition negative. For a given diagnostic test (see Zweig et al. [1993]), the true positive rate (TPR) against false positive rate (FPR) can be measured, where

$$TPR = \frac{TP}{P} = \frac{TP}{TP+FN} \text{ and } FPR = 1 - \frac{TN}{N} = \frac{FP}{FP+TN}$$

Thus, TPR is equivalent to sensitivity and FPR is equivalent to $1 - \text{ specificity } = 1 - \frac{TN}{TN+FP}$. To draw a ROC curve, only the true positive rate (TPR) and false positive rate (FPR) are needed (as functions of some classifier parameter). All possible combinations of TPR and FPR compose a ROC space. See Figure (3.17).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 3.17: The ROC space.

One TPR and one FPR together determine a single point in the ROC space, and the position of a point in the ROC space shows the tradeoff between sensitivity and specificity, that is, the increase in sensitivity is accompanied by a decrease in specificity. Thus the location of the point in the ROC space depicts whether the diagnostic classification is good or not. In an ideal situation, a point determined by both TPR and FPR yields a coordinates $(0, 1)$, or we can say that this point falls on the upper left corner of the ROC space. This ideal point indicates the diagnostic test has a sensitivity of 100% and specificity of 100%. It is also called perfect classification. Diagnostic test with 50% sensitivity and 50% specificity can be visualised on the diagonal determined by coordinate $(0, 0)$ and coordinates $(1, 0)$. Theoretically, a random guess would give a point along this diagonal. A point predicted by a diagnostic test fall into the area above the diagonal represents a good diagnostic classification, otherwise a bad prediction.
In binary classification, the class prediction for each instance is often made based on a continuous random variable X (see Appendix (24.1)), which is a score computed for the instance (such as the estimated probability in logistic regression). Given a threshold parameter T, the instance is classified as positive if $X > T$, and negative otherwise. X follows a probability density $f_1(x)$ if the instance actually belongs to class positive, and $f_0(x)$ otherwise. Therefore, the true positive rate is given by

$$TPR(T) = \int_T^{\infty} f_1(x)dx$$

and the false positive rate is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$FPR(T) = \int_T^{\infty} f_0(x)dx$$

The ROC curve plots parametrically $TPR(T)$ against $FPR(T)$ with T as the varying parameter. As an example, imagine that the blood protein levels in diseased people and healthy people are normally distributed with means of $2g/dl$ and $1g/dl$, respectively. A medical test might measure the level of a certain protein in a blood sample and classify any number above a certain threshold as indicating disease. The experimenter can adjust the threshold (black vertical line in the Figure (3.18)), which will in turn change the false positive rate. Increasing the threshold would result in fewer false positives (and more false negatives), corresponding to a leftward movement on the curve. The actual shape of the curve is determined by how much overlap the two distributions have.

Figure 3.18: The ROC Curves Characteristic.

When using normalised units, the area under the curve (AUC) is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming positive ranks higher than negative) (see Hanley et al. [1982]). This can be seen as follows: the area under the curve is given by (the integral boundaries are reversed as large

$$\begin{aligned}
TPR(T) &: T \to y(x) \\
FPR(T) &: T \to x \\
A &= \int_{x=0}^{1} TPR(FPR^{-1}(x))dx \\
&= \int_{\infty}^{-\infty} TPR(T)FPR^{'}(T)dT \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(T^{'} > T)f_1(T^{'})f_0(T)dT^{'}dT = P(X_1 > X_0)
\end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $I(\cdot)$ is the indicator function, X_1 is the score for a positive instance and X_0 is that for a negative instance, and f_0 and f_1 are probability densities. It can further be shown that the AUC is closely related to the Mann-Whitney U, which tests whether positives are ranked higher than negatives. It is also equivalent to the Wilcoxon test of ranks (see Mason et al. [2002]). For a predictor f, an unbiased estimator of its AUC can be expressed by the following Wilcoxon-Mann-Whitney statistic:

$$AUC(f) = \frac{\sum_{t_0 \in \mathcal{D}^0} \sum_{t_1 \in \mathcal{D}^1} I\big(f(t_0) < f(t_1)\big)}{|\mathcal{D}^0| \cdot |\mathcal{D}^1|}$$

where $\mathcal{D}^0$ is the set of negative examples, and $\mathcal{D}^1$ is the set of positive examples.
The AUC is related to the Gini coefficient by the formula

$$G_1 = 2AUC - 1$$

where

$$G_1 = 1 - \sum_{k=1}^{n} (X_k - X_{k-1})(Y_k + Y_{k-1})$$

Hence, it is possible to calculate the AUC by using an average of a number of trapezoidal approximations.
While the machine learning community most often uses the ROC AUC statistic for model comparison, this practice has been questioned because AUC estimates are quite noisy and suffer from other problems.

3.3.8 Representation learning

We have defined data and features in Section (3.1.1) and presented the main characteristics of data mining. Clustering and classification were discussed in Section (3.3.4) and Section (3.3.5), respectively, and some of the challenges of computational learning were introduced. However, as discussed by Bengio et al. [2013], the performances of machine learning methods for data mining are heavily dependent on the choice of data representation (or features) on which they are applied. This is a direct consequence from the objectives of machine learning discussed in Section (1.2.2) and the fact that information is discontinuous and unknown. As a result, we saw in Section (1.7) and Section (3.3.3) that a lot of effort was put in the design of preprocessing pipelines as well as data transformation to effectively represent the data.
Since the start of computing, many representation learning approaches have been proposed for learning the intrinsic structure of data. Nowadays, in machine learning, feature learning, or representation learning, is a set of techniques that allows a system to automatically discover the representations needed for feature detection, or classification, from raw data. This replaces manual feature engineering and allows a machine to both learn the features and use them to perform a specific task.

3.4 Data management

3.4.1 Overview

3.4.1.1 A brief review

Data management comprises all disciplines related to managing data as a valuable resource. With the availability of enormous amounts of data, the accessibility to historical datasets and the attainment of cloud storage, data (see Section (3.1.1.1)) became the key to the rapid growth of tech companies. All companies across all industries should realise the importance of using data to increase their value. Data is the most valuable asset for a company. It can be used for customer analytics, taking business decisions that can have an impact on topline, providing better user experience to address the changing customer behaviour. Data analytics help the company understand its customers' behaviour and

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

preferences. It can also help identifying new segments of business by creating targeted products. More generally, data should be used to improve customer experience, reduce risk, increase efficiency, grow business. The recent explosion of data, mainly due to the scaling of remote work and particularly the rapid deployment and development of smart work from home technologies, is pushing AI and machine learning to combine with existing technologies such as data management, cybersecurity, IoT, automation, IT operations. In all cases, it has resulted in the enhancement of these technologies in terms of size, speed, productivity, cost.

Data quality is defined as the level of data fitness for a given purpose by indicating the degree to which the data is complete, consistent, free from duplication, accurate and timely for that purpose. Data quality management is the application of relevant practices and controls to improve data quality. In general, the assessment of data quality is complicated by the fact that it is captured in one context and used in totally different contexts. As a result, it is domain-specific and requires significant human involvement for capturing and maintaining accurate data. We saw in Section (3.1.1) that incomplete and missing data, inaccurate data, dated data, and duplicate data were the most common data quality problems. This led to a prolific literature on data quality management (see Sadiq et al. [2011]). As discussed in Sections (1.6) (1.7), AI and machine learning require the availability of coherent data with appropriate computing infrastructure. Further, we saw in Section (3.2.4.2) that operational data (business transactions) was structured and stored in data warehouses, where the data was cleaned, transformed to create a comprehensive database and then integrated. The Extract, Transform, and Load (ETL) tools are used to ease the construction of data warehouses by ensuring data quality at the source. However, with the recent explosion of data, and since the data is integrated from multiple sources, data quality management becomes critical.

Researchers address the problem of missing data, identification of duplicate data, linking and integrating related data obtained from multiple sources by employing algorithmic approaches based on rules and statistical methods (see Maydanchik [2007], Osborne [2013]). When it comes to assessing data quality for information systems, one need to consider the contribution of user interfaces (see McGilvray [2008]). One can also assess the quality of web data sources by considering either hyperlinks and browsing history (see Dong et al. [2015]), or the relationships between the web sources (see Yin et al. [2008]). With the recent data explosion, streaming, data heterogeneity, and cloud deployments are creating new challenges (see Cheah [2014], Gudivada et al. [2015]). Systems for data management called NoSQL have been introduced to address the storage and retrieval needs of big data applications (see Gudivada et al. [2014]). However, in those systems, performance at scale takes precedence over data quality, favouring real-time analysis for the operational and strategic decision making process. This has led to machine learning predictive algorithms to develop across all industries. However, those models are extremely sensitive to the quality of the data, as poor data can lead to biased and erroneous results.

In machine learning, training data contains various errors that need to be detected and fixed (see Section (3.3.3)). For instance, data quality issues maybe due to data entry errors, inconsistent format of incoming data, the data maybe dirty, the data may change as the data source itself evolves, the data maybe missing or incomplete due to possible bugs in the data source. Further, data is becoming less structured and it is physically no-longer in-house as hybrid cloud data architecture develop. Data quality has a profound impact on model accuracy, hence the need for improving and controlling data quality. As discussed in Section (1.6), the learning process consists of model representation, accuracy measures for assessing performance, and optimisation for identifying the best model in the model space using the validation data. During this learning process, different data subsets are used, which must be accounted for when assessing the quality of data for ML models. Poor data quality such as missing data, duplicate data, highly correlated variables, large number of variables, outliers pose significant problems when building ML models. For instance, outliers in training dataset can cause instability and non-convergence in the training. Incomplete, inconsistent, and missing data can lead to drastic degradation in prediction. Pre-processing operations on the data helps these models achieve increased effectiveness. Probabilistic and statistical methods for missing data imputation, outlier detection, data transformations, dimensionality reduction, robust statistics, cross-validation and bootstrapping are critical for ML models (see Gudivada et al. [2017]).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

So far, the existing data remediation strategies have been manual, reactive, and consume a lot of time and effort. The impact of data quality issues is both internal and external to a company. In the former, data quality issues can break the trust required from other units to use the data in forecasting and reporting. Additionally, regulatory reporting and reputation of the company can be affected due to poor data quality. With the development of machine learning, we can now apply these algorithms to perform and replace statistical analysis. For instance, natural language understanding, language modelling, and named entity recognition can be used for solving some of the data quality issues (entity resolution see Talburt [2013]). Development in visual analytics can be used for outlier detection (see Venna et al. [2016]) to move to a semi-automated process. Big data driven ML is now yielding solutions achieving automatic domain adaptation through supervised and unsupervised learning (see Gudivada et al. [2015b]). While data quality errors are often spotted in the transformed data, new algorithms are developing to identify the original data elements and their sources corresponding to these errors (see Chalamalla et al. [2014]).

3.4.1.2 Some challenges

We present a few challenges related to data quality management.

3.4.1.2.1 Missing data From a statistical perspective, missing data can be classified into three categories (see Allison [2001])

1. Missing completely at random (MCAR): There is no pattern to the missing data. Data is missing independently of both observed and unobserved data. The missing and observed values have similar distributions. That is, MCAR is just a subset of the data.

2. Missing at random (MAR): Given the observed data, data is missing independently of unobserved data. It is possible that there are systematic differences between the missing and observed values that can entirely be explained by other observed variables.

3. Missing not at random (MNAR): Missing observations are related to the unobserved data. Therefore, observed data is a biased sample.

In the case where both MCAR and MAR prevail for a variable, such variable can be dropped across all observations in the dataset. On the other hand, dropping MNAR variables may lead to strongly biased results. Among the different approaches for dealing with missing data, the simplest one consist in deleting all observations from the dataset that have missing values. However, if a large proportion of observations have missing values for a critical attribute (or feature), its deletion will have a significant impact when training ML models. Alternatively, one can perform pairwise deletion. For example, assuming a dataset with three variables v_i, $i = 1, 2, 3$, with some observations having missing values for the feature v_2. The statistical method must be such that it does use v_2. Any statistics on v_1 and v_3 is possible. Another approach is to use mean substitution, where the mean is computed for a group of observations or the entire dataset. One can also use multiple regression on a set of highly correlated variables, but it may lead to overfitting for ML models. Multiple imputation approach can also be used for predicting missing values. Methods such as expectation-maximisation (EM) / maximum likelihood estimation, Markov Chain Monte Carlo (MCMC) simulation, and propensity score estimation are used to estimate the missing values. A version of the dataset corresponding to each method is created, the datasets are analysed, and the results are combined to produce estimates and confidence intervals for the missing values.

3.4.1.2.2 Duplicate data Duplicates are ubiquitous, especially in user-contributed data in social network applications (a user unintentionally create a new profile when it already exists). There are two major issues to identifying duplicate data in the context of big data:

1. Linking: Assigning a unique identifier to various pieces of information belonging to the same entity. The unique identifier is used to aggregate all information about the entity.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. Identifying and eliminating duplicate data based on the unique identifiers. It is resource-challenged as the data is too big to fit in main memory all at once. Bloom filter is a solution requiring that the associated hash functions be independent and uniformly distributed. It is a space-efficient probability data structure for testing set membership.

3.4.1.2.3 Data variety Among the five Vs characterising big data (see Section (3.2.4)), variety (data heterogeneity) poses great challenges to data quality. With the emergence of Web data, new research is developing on tackling data quality issues associated with semi-structure data in Web pages. One must refer on information extraction (IE), which synthesise structured information such as entities, attributes of entities, and relationships among the entities from unstructured sources. IE systems evolved from manually coded rule-based systems, generative models based on Hidden Markov Models (HMM), conditional models based on maximum entropy, to hybrid models that leverage both statistical and rule-based methods. Some IE research focuses textual data, while other research consider extracting text present in images and videos. Some work is done on extracting information from images and video. IE data quality challenges come from the uncertainty associated with the extracted information.

3.4.1.2.4 Semantic data integration When the data is structured and incomplete, we need to identify and integrate related data to provide users a comprehensive, unified view of data. However, integrating unstructured heterogeneous data remains a significant challenge. Practical cases of data quality issues related to information extraction and data integration are seen with the development of smart cities.

3.4.1.2.5 High-dimensional data space The curse of dimensionality refers to various phenomena that arise when analysing and organising data in high-dimensional spaces that do not occur in low-dimensional settings (see Bellman [1957]). When the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method requiring statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality. Further, organising and searching data often relies on detecting areas where objects form groups with similar properties. However, in high dimensional data, all objects appear to be sparse and dissimilar in many ways, which prevents common data organisation strategies from being efficient. Thus, irrespective of the size of the training data, it only covers a small fraction of the input space, so that no dataset is big enough to compensate for dimensionality curse. For example, in machine learning problems that involve learning a state-of-nature (such as reinforcement learning) from a finite number of data samples in a high-dimensional feature space with each feature having a range of possible values, then an enormous amount of training data is required to ensure that there are several samples with each combination of values. When it comes to predictive performance, the curse of dimensionality is used interchangeably with the peaking phenomenon (also known as Hughes phenomenon), which states that with a fixed number of training samples, the average (expected) predictive power of a classifier, or regressor, first increases as the number of dimensions or features used is increased but beyond a certain dimensionality it starts deteriorating instead of improving steadily (see Trunk [1979], McLachlan [2004]).

One solution is to use variable/feature reduction techniques. For instance, variable selection (or feature selection) is the process of choosing a minimal subset of relevant variables that are maximally effective at model building. Examples of techniques for dimensionality reduction include principal component analysis (PCA) and exploratory factor analysis (EFA). Another solution is to realise that common-sense heuristics based on the most straightforward methods can yield results which are almost surely optimal for high-dimensional problems (see Kainen [1997], Donoho [2000]). This is called blessing of dimensionality. Both the blessing of dimensionality and the curse of dimensionality are two sides of the same coin (see Gorban [2018]). For example, the typical property of essentially high-dimensional probability distributions in a high-dimensional space is: the squared distance of random points to a selected point is, with high probability, close to the average (or median) squared distance. This property significantly simplifies the expected geometry of data and indexing of high-dimensional data (blessing), but, at the same time, it makes the similarity search in high dimensions difficult and even useless (curse).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.4.1.2.6 Data quality dimensions Data quality is often measured as a function of a set of dimensions which provides a basis to measure and monitor the quality of data. There are several dimensions proposed in the literature. Some of these dimensions include

- accuracy
- currency
- consistency

The initial assessment of data quality forms the baseline. Enhanced data quality resulting from new data cleaning, transformation, integration, and aggregation activities is measured against the baseline. When it comes to dealing with big data, one need generic data quality dimensions which transcend application domains. Some of these dimensions can be evaluated by using an ordinal scale. Some examples with their description are as follows (see Gudivada et al. [2017]):

- Data governance dimension: Each question in the description column corresponding to this dimension is regarded as a sub-dimension. The latter can be measured using an ordinal scale such as

  $$\{ \text{No data standards exist, Data standards exist but are not enforced, Data standards exist and are enforeced} \}$$

  Is there data standards within the organisation and are they enforced? Do clearly defined roles and responsibilities exist for data quality related activities?
- Data specifications: Are data standards documented in the form of a data dictionary, data models, meta data, and integrity constraints?
- Data integrity: How is data integrity maintained? How are data integrity violations detected and resolved?
- Data consistency: If data redundancy exists, how is data consistency achieved? What methods are used to bring consistency to data that has become inconsistent? If data is geographically replicated, how is the consistency and latency managed?
- Data Currency: Is the data current? Do procedure exists to keep the data current and purge stale data?
- Data duplication: It can be quantified numerically. For example the ratio of the number of duplicate observations to the total number of observations.
- Data completeness: Is the data about entities complete? How is missing data managed?
- Data provenance: Is a historical record of data and its origination maintained? If the data is acquired through multiple sources and has undergone cleaning and transformations, does the organisations maintain a history of all the changes to the data?
- Data heterogeneity: If multi-modality data about an entity is available, is that data captured and used?
- Streaming data: How is streaming data sampled, filtered, stored, and managed for both real-time and batch processing?
- Outliers: It requires sophisticated methods to quantify it. Are outliers detected and addressed? Are there versions of datasets that are outlier-free? Does each version correspond to a different method for outlier detection and treatment?
- Feature selection: Do datasets have versions excluding features that are either redundant, highly correlated, or irrelevant? How many versions are available?

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Feature extraction: Do the datasets provide a set of derived features that are informative and non-redundant, in addition to the original set of variables/features? How many such derived feature sets are available?
- Business rules: Does a process exist to identify, refine, consolidate, and maintain business rules that pertain to data quality? Do rules exist to govern data cleaning and transformation, and integrating related data of an entity from multiple sources? What business rules govern substitutions for missing data, deleting duplicate data, and archiving historical data? Are there rules for internal data audit and regulatory compliance?
- Data accuracy: Data can be syntactically accurate and yet semantically inaccurate. For example, a customer's mailing address can meet all the syntactic patterns specified by the postal service, yet it can be inaccurate. How does the organisation establish the accuracy of data?

3.4.2 Data governance

Data governance is a set of best practices and controls set within an enterprise to actively manage and improve data quality. It provides a process-oriented framework to embed and execute data quality activities such as planning, cleaning, profiling, assessing, issue tracking, and monitoring. It identifies clear roles and responsibilities for ensuring data quality through repeatable processes. Overall, it provides the enterprise with a proactive and holistic approach to data quality by accurately capturing data and enforcing controls to prevent deterioration of data quality. Data governance calls for establishing a data governance strategy, policies, procedures, roles and responsibilities.

3.4.2.1 Data quality lifecycle

One first need to understand the data quality lifecycle in the data governance environment in order to suitably identify the procedures for assessing the various data quality dimensions and to implement the defined procedures. The lifecycle depicts the movement of data through the various processes and system in the enterprise. This is illustrated in Figure (3.19) coming from Gudivada et al. [2017].

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 3.19: Data quality lifecycle in data governance.

The different components of the data quality lifecycle are:

1. Data governance standards:
   - Data dictionary (or glossary): It is a repository of definitions of business and technical terms including naming conventions, meaning of data, its origin, owner, usage, and format. It can include relationships of a data item to other data elements, default values, minimum and maximum values. It can also include functions provided by taxonomies and ontologies.
   - Reference and meta data:
     - Reference data is any data that can be used to organise, classify, and validate other data. For example, codes for airports, zip codes, classification of instructional programs (CIP) codes. Some data entered by application users can be validated against reference datasets.
     - Master data represents core business data such as products, services, customers, and suppliers. That is, entities that come into play in business transactions such as customer placing an order for a product.
     - Meta data refers to additional data about data whose scope is beyond the data dictionary. It is typically generated by other processes in the data quality framework.
   - Data models and business rules: They documents a range of data models and policies.
     - Data models include the relational data model, NoSQL data models such as column-family and graph models.
     - Business rules specify complex relationships between data elements from a validation perspective. They also include rules for missing data imputation, spotting data integrity violation, resolving inconsistent data, linking related data, detection of outliers, and purging dated data, among others.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Roles and responsibilities: It involves identifying various roles for data quality management and associating responsibilities with each role.
  - Roles may include an owner, program manager, project leader, chief data officer, business analyst, data analyst, and data steward. For instance, the data analyst may address issues raised in data quality reports and track resolution of these issues.
  - Responsibilities depend on the organisation.

2. Data quality analytics: It consists of four categories of analytic functions providing the algorithms and statistical methods required for the implementation of Data acquisition and semantics generation, TIA processes and semantics generation, Data quality assessment, and Data quality monitoring. These analytic functions are:
   - Descriptive analytics: It provides a process and a set of tools for measuring and assessing data quality. It aims at describing data quality both quantitatively and qualitatively using statistical exploratory data analysis techniques.
   - Diagnostic analytics: It determines the factors contributing to poor data quality by assessing the relative contributions of factors such as missing, incomplete, inconsistent, and duplicate data.
   - Predictive analytics: It helps answering the what-if questions related to data quality improvement. For example what would be the magnitude of data quality improvement if 90% of incomplete data was resolved.
   - Prescriptive analytics: It suggest an actionable plan for solving the predictive analytics.
3. Data acquisition and semantics generation: It consists of three processes
   - Filtering and sampling: We saw in Section (3.2.4) that big data applications were acquired from multiple sources. To manage the complexities of velocity and volume facets, streaming data is sampled and filtered. Data is randomly sampled for retention and further processing. Further, the data can be averaged over a moving window and retained in that format. Data can be filtered based on some predicates. For example, if the values fall within a predefined range, it is then retained, or alternatively, data falling outside the range is retained. Such data represents anomalies and outliers related to unusual events.
   - Cleaning and enrichment: It uses analytic functions to automatically or semi-automatically perform data standardisation, detection and removal of duplicates, inconsistent data resolution, imputation for missing data, incomplete data strategies, and matching data across multiple sources. For instance, annotation is a meta data by-product of cleaning and enrichment, which is used to explain data evolution. Some cleaning procedures use a range of data transformations such as data normalisation. One can automatically determine missing values if the missing value in conjunction with other available values and associated constraints entail a unique solution. Some methods for cleaning data can also be based on detecting outliers, or on imputation for estimating missing values (see Section (3.4.3)).
   - Semantics and meta data generation: Some additional pieces of information are also captured during the data acquisition phase. For example, in the case of IoT, one also records the types of sensors used, precision and scale for data calibration, and sampling methods. In general, these semantics and meta data are essential for determining suitable statistical analysis methods in downstream big data and machine learning applications.
4. Transformation, Integration, Aggregation (TIA) processes and semantics generation:
   - Transformation: In general, statistical techniques assume data normality and equality of variance. However, data may be noisy with non-constant variance. Thus, data must be transformed to improve data normality and equalise variance. This includes inversion and reflection, conversion to a logarithmic scale, and square root and trigonometric transforms.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Integration and Aggregation: Since large companies have very complex data environments comprising several data sources, one must integrate and aggregate data from these sources. Integration requires accurately identifying the data associated with an entity originating from multiple sources. Unfortunately, the data source providers do not use consistently the same identification schemes. There are several tools available for data integration such as the Talend Open Studio and Pentaho Data Integration. Once the data is integrated, it must be aggregated. Aggregates are usually precomputed to speed up the analysis tasks.
- Semantics generation: The TIA processes generate substantial meta data as well as semantics, which are also captured and stored.

5. Data storage, retrieval and purging: It provides persistent storage, query mechanisms, and management functionality to secure and retrieve data. In general, one uses both relational and NoSQL database systems to realise this functionality. Several data models and query languages are provided to efficiently store and query structured, semi-structured, and unstructured data. Rule-driven processing is used to purge expired data and stale data.

6. Data quality measurement and assessment: There are several primary functions

   - Dashboards view of current data quality: Dashboards are graphical depictions of data quality along and across the dimensions. Measures and metrics are used to quantify these dimensions.
   - Measures and metrics: Measures quantify more concrete and objective attributes such as the number of missing values, while metrics quantify abstract, higher-level, and subjective attributes such as data accuracy. The measurement scale determines which statistical methods are suitable for data analysis. Some of these measurement scales are:
     - The nominal scale: It only provides a set of labels for measurements. There is no inherent ordering to the labels. For example, gender is measured on a nominal scale comprised of two labels: male and female.
     - The ordinal scale: It is a nominal scale when ordering is imposed on the labels. For example, the Mohs scale of mineral hardness varies from 1 to 10, where 1 corresponds to talc and 10 to diamond, but it is not uniform.
     - The interval scale: The scale has equal distance between values, but it does not have an inherent zero. For example, a temperature is measured using an interval scale, but 0 degrees Fahrenheit does not represent the complete absence of temperature.
     - A ratio scale: It is an interval scale with an inherent zero. The ratio of two values on the ratio scale is meaningful. For example, money is measured on a ratio scale.
   - Data quality assessment: It is the process of evaluating data quality to identify errors and discern their implications. The assessment is made in the context of an intended use and suitability of data for that use.
     - Data profiling: It is an exploratory approach to data quality analysis. Statistical methods are used to reveal data usage patterns as well as patterns in the data (see Osborne [2013]). Several tools exists for data quality assessment using data profiling such as Tableau and Talend Open Studio.

7. Data quality monitoring: Its main function is to continually monitor data quality and report results through dashboards and alerts. In addition to the fact that big data comprising several data sources, it is acquired through batch and real-time feeds, problems arise when different values (multiple vendors) are reported for the same data item. For example, financial companies obtain real-time price information on a product from multiple vendors. If the time-stamped price received from vendors differ, the system must make a choice for selecting the right price. One solution is to have a hierarchy of vendors, ordering them based on some criteria. Muller et al. [2012] discussed how to improve data quality using the conflicting data values about the same object originating from multiple sources. Another source of data quality degradation is when systems are merged or upgraded as they may be operating under different sets of business rules and user interfaces. Further, the data

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

may overlap between the systems, and the overlapping data may be contradictory. Systems merging require substantial manual effort and decisions made in resolving data problems must be documented as meta data. TIA are also sources of data degradation as they may entail loss of data precision and scale.

3.4.2.2 Data management function

Data management function (DMF) is a core set of business processes, such as Finance, Human Resources, or Facilities Management, that provides resources to facilitate the establishment and adoption of best practices across data management disciplines, and which will always be needed across the full lifecycle of the data. It provides data management resources to ensure that the data is managed effectively as a organisational asset, providing oversight and support to establish and maintain data management best practices, facilitate adoption, and ensure utilisation. In an organisation, the lifecycle of data starts at the source and evolves before reaching the consumers (see Figure (3.19)). Some of the steps of DMF include:

- Data acquisition: (1) one need to finalise data consumption requirements, (2) prioritise data sources to meet step (1), (3) standardise the source-to-target data acquisition process by defining a common data model.
- Data maintenance: (1) build and maintain a global warehouse for data aggregation and data distribution, (2) ensure data integrity through data governance and master data management, (3) implement exception handling and data remediation based on pre-defined business rules.
- Data consumption: (1) standardise the warehouse-to-consumer data delivery process, (2) continuously monitor timeliness and quality of data delivery.

As discussed in Section (3.1.1), the data management function is responsible for getting clean data driven by data quality such as accuracy, consistency, completeness, timeliness, integrity, conformity, veracity available to consumers. Some of the activities performed to manage data include:

- Simple technological efforts to prevent faulty data from reaching the data consumers such as data validation, data enrichment, and data transformation.
- Data quality and business rules mapped to the business policies.
- Identifying the critical data elements (CDEs): The elements determined to be vital to the successful operation of the organisation.
- Service level agreements (SLAs) [2] to receive incoming data and send outgoing data.
- Data quality platforms producing data quality scorecards, which helps get a view on data source, and quantify the quality of data. These metrics are used to create an impact taxonomy by tracing the data quality issues to the undesired financial impact they may have on the business.

3.4.2.3 A reference architecture

Gudivada et al. [2017] proposed a reference architecture for implementing the data governance framework for data quality lifecycle. This is displayed in Figure (3.20).

[2] It is a commitment between a service provider and a client. Particular aspects of the service such as quality, availability, responsibilities, are agreed between the service provider and the service user.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 3.20: Data governance framework for data quality lifecycle.

The layers of the architecture (from bottom to top) are detailed as follows:

- All the types of data managed by relational and NoSQL systems as well as application data, business rules, reference and master data, and meta data are stored in the bottom layer.
- Above, we get the hardware layer powered by both conventional CPUs and processors specifically designed for ML tasks such as the neuromorphic chips.
- Next, the parallel and distributed computing layer contains software abstractions, database engines, and open source libraries to provide performance at scale for big data and ML environments.
- The, we can perform the four types of data analytics: descriptive, diagnostic, predictive, and prescriptive. These data analytics functions are performed with statistical and probabilistic models as well as with ML algorithms.
- Then, the data access layer is responsible for enforcing the data governance methods for implementing data quality. It uses a rules engine to execute and manage business rules. It controls encryption, security, privacy, compression, and provenance aspects of the data.
- The next layer provides authorisation and access control features. It also features privileged functions for system administrators.
- At last, this architecture specifies several types of query APIs for applications, interactive users, and system administrators.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.4.3 Probabilistic and statistical models

While data cleaning originally focussed on name and address validation, it expanded to standardisation of fields (such as canonical representation for date values), validating values using regular expressions, parsing and information extraction, rule-based data transformations, linking related data, merging and consolidating data from multiple sources, and elimination of duplicates. Some tools help fixing incomplete and missing data via statistical imputation. However, some tasks still remain manual, such as the data accuracy dimension. We are briefly going to discuss some of these statistical tools.

3.4.3.1 Data imputation

Missing data can causes several problems such as introducing a substantial amount of bias, making the handling and analysis of the data more arduous, and creating reductions in efficiency. One can use data imputation for estimating the missing values. In statistics, imputation is the process of replacing missing data with substituted values. When substituting for a data point, it is known as unit imputation, and when substituting for a component of a data point, it is known as item imputation. For example, imputing missing values without considering edit restrictions and then apply minimal adjustments to the imputed values to comply with the edit restrictions. Once all missing values have been imputed, the dataset can then be analysed using standard techniques for complete data. However, imputation results in datasets with very different distributional characteristics compared to the original dataset (see Finch [2012]).
There are several methods to account for missing data but the majority of them introduce bias. These methods include: hot deck and cold deck imputation; listwise and pairwise deletion; mean imputation; non-negative matrix factorisation; regression imputation; last observation carried forward; stochastic imputation; and multiple imputation. We briefly describe some of these methods:

- The imputation of the mean: In this method, the mean is computed over the observed values and used to replace any missing value. As a result, the sample mean for that variable is not modified. However, it causes a bias in measures of spread, which are estimated from the sample after imputation. It can be carried out within classes, and can be expressed as $\widehat{y}_i = \overline{y}_h$ where $\widehat{y}_i$ is the imputed value for record i and $\overline{y}_h$ is the sample mean of respondent data within some class h. This is a special case of the generalised regression imputation

$$\widehat{y}_{m,i} = b_{r,0} + \sum_j b_{r,j} z_{m,ij} + \widehat{\epsilon}_{m,i}$$

  where $b_{r,0}, b_{r,j}$ are estimated from regressing y on x in non-imputed data, z is a dummy variable for class membership, and data is split into respondent (r) and missing (m) (see Kalton [1986]).

- The ratio imputation model: The imputation value of variable x_i is computed as the product of $\widehat{R}y_i$, where y_i is a covariate of x_i and $\widehat{R}$ is an estimate of the average ratio between x and y. $\widehat{R}$ is computed as the ratio of the sum of observed x values to the sum of corresponding y values.

- The generalised linear regression model: The missing value $\widehat{x}_i$ is imputed as

$$\widehat{x}_i = \widehat{\beta}_0 + \widehat{\beta}_1 y_{1,i} + \cdots + +\widehat{\beta}_k y_{k,i}$$

  where β_l, $l = 0, 1, ..., k$ are estimated coefficients of the auxiliary variables y_l, $l = 0, 1, ..., k$.

- The hot deck: In the hot deck imputation method, the missing values are imputed by copying values from similar records in the same dataset. This method can be applied to both numerical and categorical data. The main consideration is determining which one of the observed values should be chosen for imputation. It led to several variations such as random hot-deck imputation, sequential hot deck imputation, and predictive mean matching.

- The k-nearest neighbour imputation

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Imputation of missing longitudinal data: An array of methods for imputation of missing longitudinal data (see Engels et al. [2003].
- Multiple imputation: n order to deal with the problem of increased noise due to imputation, Rubin [1987] developed a method for averaging the outcomes across multiple imputed data sets to account for this. All multiple imputation methods follow three steps.

  1. Imputation: Similar to single imputation, missing values are imputed. However, the imputed values are drawn m times from a distribution rather than just once. At the end of this step, there should be m completed datasets.
  2. Analysis: Each of the m datasets is analysed. At the end of this step there should be m analyses.
  3. Pooling: The m results are consolidated into one result by calculating the mean, variance, and confidence interval of the variable of concern or by combining simulations from each separate model.

  However, the primary method of multiple imputation is multiple imputation by chained equations (MICE). It has been show to work very well on missing at random data, though there is evidence to suggest, through a simulation study, that with either a sufficient number of auxiliary variables it can also work on data that is missing not at random; use of a latent variable (derived through a Latent Class Analysis method produces more accurate estimates over MICE) (see Section (11.5)).

3.4.3.2 Outliers

Several solutions exist for detecting data quality problems, especially outlier detection. Outlier data is so different from the rest of the dataset that it can skew models and statistical measures. One can use probabilistic and statistical models, linear correlation analysis, proximity-based detection, supervised outlier detection, among others, to detect such outliers. When considering probabilistic and statistical models, one can use normal distribution and Zipf distribution. For instance, statistical process control (SPC) (see Sessions et al. [2006], DeVor et al. [2007]) is a procedure of quality control based on statistical methods. Given the characteristics of the normal distribution in Section (11.1.2.2) It uses control charts [3] containing a center line (CL) as well as lower (R_{LCL}) and upper (R_{UCL}) control limits given by

$$R_{UCL} = \mu + 3\sigma \text{ and } R_{LCL} = \mu - 3\sigma$$

where μ is the population mean, and σ is the population standard deviation.
So far, data quality control methods are mostly based on users' experience and previously established business rules, which is time consuming, limits performance and provides low accuracy. Popular data quality software tools contain profiling modules based on simple statistical algorithms (see Michalski et al. [2013]). Fortunately, one can use machine learning techniques to leverage computing resources, providing greater value to users. For instance, O'hagan et al. [2015] discussed data quality assessment architecture via AI. Dai et al. [2015] proposed a data quality architecture to identify data quality and improve it with Deep Learning and statistical quality control models. The deep learning model (six layers backpropagation network) is used for prediction (P) of employee annual salaries from the Arkansas state human resource department, while the statistical quality control module calculates the errors between the predicted value and the real (actual) value predicted (T). If the difference rate is out of the range $[R_{LCL}, R_{UCL}]$ from the quality control chart, the data will be marked as outlier data, or risk data.

3.4.3.3 Duplicate detection

Duplicate detection (DD), or record linkage, consists in identifying objects (pairs of attributes/records) in databases that refer to the same real world entity. Objects may be duplicates, even though they are not identical due to errors

[3] A graph or chart with limit lines.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

and missing data. Errors are introduced as the result of transcription errors, incomplete information, lack of standard formats, or any combination of these factors. Common approaches to duplicate detection use either rules or a weighted aggregation of similarity measures between the individual attributes of potential duplicates. However, choosing the appropriate rules, similarity functions, weights, and thresholds requires deep understanding of the application domain or a good representative training set for supervised learning approaches. This is discussed in Section (3.5). Elmagarmid et al. [2007] presented a thorough analysis of the literature on duplicate record detection. Apart from graph databases, there are three main approaches used for duplicate detection:

1. iterative algorithms, which classify one pair of candidates at every iteration.
2. the use of clustering techniques.
3. machine learning, where models and similarity measures are learned.

Lehti [2006] proposed an unsupervised, domain independent approach to duplicate detection that starts with a broad alignment of potential duplicates, and analyses the distribution of observed similarity values among these potential duplicates and among representative sample non-duplicates to improve the initial alignment. These distributions are used in a modified Fellegi-Sunter model to iteratively remove clear non-duplicates from the set of potential duplicates. When using supervised learning, we require a large amount of manually labelled data to train the models on. To improve the accuracy of these models one can use word embedding and word hashing as extra features, With semi-supervised deep learning techniques, we allow the training to use both labelled data and unlabelled data, which is easily available. Following this approach, Janssen [2016] assessed the accuracy of deep belief networks against classifiers, showing some differences. Santos et al. [2018] proposed a string matching approach using deep neural networks to classify pairs of toponyms as either matching or non-matching. Verissimo de Sousa Marinho [2018] extended this approach with neural architectures integrating shortcut connections, self attention mechanisms, hard alignment attention between the strings being compared, or max-pooling over the sequences of outputs of the recurrent layers, either considering a single RNN or a hierarchy of RNN layers. Ranganathana et al. [2018] focused on improving the accuracy of the duplicate record detection process, along with other well-known measures, by adopting different optimisation algorithms and getting the optimal threshold automatically to find the duplicate records more accurately minimising human intervention. Recently, some literature developed around graph theory for detecting duplicates by using relationships among object representations to improve results. This is called duplicate detection in graph data (DDG). When it comes to applying methods to very large data sets, it is essential to scale not only in time but also to scale in space, for which relational databases are commonly used.

3.4.4 ML in data management

3.4.4.1 Some examples

One can use ML in data management function for detecting anomalies and prescribing remedies to improve error-detection. Raghavan [2020] presented some example cases:

- Data cleaning: We can define a scoring mechanism to specify a likelihood to the data values inserted in user interfaces. We can then edit the data items with low confidence score at a later stage to increase their likelihood. Provenance tracking should be tightly integrated with this method. We can also leverage ML algorithms to suggest appropriate data values. For example, these values could be suggested based on an underlying statistical model of data distribution (see Section (11.5)).
- Automated data entry: Rather than manually entering the data to the different systems, one can use OCR and speech-to-text to scan documents of any type and convert recordings of human speech to text, respectively. For example, relevant unstructured data can be extracted from documents using ML algorithms, which can then be automatically updated to individual applications.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Improve data quality at the source: ML can be used to resolve common errors during manual data entry such as incorrect spellings, incomplete addresses. Supervised ML algorithms can continuously learn against the reference data to improve the accuracy of the predictions. If the reference data is not available, record linkage can be used to link the different data sources and identify the closest match to predict values.

- Fill data gaps: In general, the features of a record are correlated in one way or another. For example, if the product code is one of the attributes, they are usually related to the product identifier, country, and region among other features. When there is a relationship between the missing data and the other features of the record, or when there is historical information on the probable data values, supervised ML algorithms can be used to fix the data issues by predicting the values to correct the missing values. When historical information is not available, expectation maximisation algorithms can be used to find the maximum-likelihood estimates. Feedback can help the algorithms to learn and build accuracy over time.

- Improve incorrect reporting: During regulatory reporting, incorrect records that should not be reported may be inadvertently reported to the regulators. ML algorithms, such as random forest and logit model, can be used to identify and remove these records from the report, improving the quality of regulatory reporting.

- Monitor load and SLAs: Feeds from source systems can be daily, weekly or monthly and vary by regions and products. SLAs govern both the acquisition of source data and supply of aggregated data to the consumers. Supervised ML, such as regression and classification, can be used on historical data on volume and SLAs to predict the load in an incoming feed or identify potential breaches in SLAs in advance.

- Create business rules: Supervised ML algorithms can be used to learn from any existing business rules engine and use the data from the warehouse to both create new business rules and improve upon the existing ones to assist in building a robust rules engine framework. For example, a business rule may define the loss threshold at $\$1M$. The total operational loss of the business unit may be determined based on the number of losses above $\$1M$, which also has an impact on the operational risk capital as it ignores all the losses below $\$1M$. The ML algorithms can help resolve this issue by categorising loss events into different buckets of varying degrees (bucket 1 in the range $[100, 200]K$, bucket 2 in the range $[200, 500]K$, bucket 3 in the range $[500, 1000]K$) based on both frequency and size to determine the total loss that will help the risk team to identify the thresholds more accurately.

- Reconciliation: Using historical data and user actions that may have resolved reconciliation issues in the past, ML algorithms can learn about exceptions management during reconciliation, resolving the issues more effectively.

3.4.5 Graph database

In general, iterative duplicate detection in graph data (DDG) algorithms consist of the following steps:

- Initialisation: A graph representation of the considered data is created, precomputations are performed, and a priority queue of pairs of candidates (object representations among which duplicates should be detected) is set up.

- Iterative phase: Pairs of candidates are retrieved from a priority queue, and then classified as duplicates or non-duplicates, and potentially trigger some update in the priority queue. The most significant update is to sort the priority queue when a duplicate was classified. This is done to reduce the number of classifications performed for a candidate pair (which can be more than one in DDG).

Step $[2]$ is described in Figure (3.21).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 3.21: DDG iteration.

A pair of actor candidates $(a_1, a_1^{'})$ is retrieved from the priority queue residing in main memory. As it is classified as a duplicate, the movie candidate $(m_1, m_1^{'})$ pair rises in the order of the priority queue, because its similarity increases based on the fact that actor a_1 playing in m_1 is the same as actor $a_1^{'}$ playing in $m_1^{'}$.
Herschel et al. [2008] Weis et al. [2008] proposed to scale up the iterative phase of DDG by scale up the above individual steps. They assumed that all necessary data structures, in particular the graph and the priority queue are stored in a relational database. This is summarised in Figure (3.22).

Figure 3.22: DDG iteration scale up.

3.5 Duplicates detection

3.5.1 Introduction

3.5.1.1 Description of the problem

Duplicate detection aims at finding objects from various data sources that refer to the same real world entity, where these individual objects [4] might be erroneous and incomplete. This problem has been addressed by the statistics community as record linkage, the database community as deduplication, the ML community as identity uncertainty. For example, in data integration (see Section (3.2.4.2)), one is concerned by linking in relational terms, two or more tables on their key fields. However, the data often lack a unique global identifier that would permit such an operation. We focus on database records and consider the record linkage, or record matching problem, where we want to identify records in the same or different databases that refer to the same real-world entity, even if the records are not identical.

The duplicate detection problem can be formalised as follows: Given two sets of objects A and B, we want to divide

[4] An object is a vector of attributes (or fields, properties, features).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

the set of all object-pairs [5] $(a, b) \in A \times B$ into a set of matching object-pairs M and unmatching object-pairs U. In order to discover matching entries in tables A and B, we can execute a nested-loop comparison by comparing every record of table A with every record in table B. However, such a strategy requires a total of $|A| \cdot |B|$ comparisons, a cost that is prohibitively expensive even for moderately sized tables. The problem is further complexified as there are numerous possible differences when comparing the individual attribute-pairs. Some examples include:

- Typographical errors: The same string contains wrong, missing, additional or interchanged characters or tokens. Similarity measures for this kind of errors are numerous, such as distance functions, measuring dissimilarity rather than similarity.
- Toponym matching: This is the problem of matching place names that share a common referent. It is used to address the ambiguity problems related to the fact that alternative names can be used to refer to the same place. For example, the names **Lisboa** and **Olissipona** can both be used to refer to Lisbon, the capital city of Portugal.
- Datatype dependency: If a value is not a simple string, but some kind of primitive datatype, simple string-based similarity measures may poorly work. For example, a year value of 2000 and 1999 do not have a single character in common and would therefore be judged as very different by a string-based similarity measure. On the other hand, a simple arithmetic difference based similarity measure would be much more effective. However, for the same datatype year the values 2001 and 2010 would be better judged by the string-based similarity, as a simple typo is possible.
- Domain dependency: This is when an equivalence between values can only be recognised if additional domain knowledge is available. For example, in the publication domain the two values for the conference event **VLDB-95** and **Int. Conference on Very Large Data Bases, 1995** look very different and can only be recognised as equivalent, when knowing that **VLDB** is a common abbreviation for **Int. Conference on Very Large Data Bases**.

Thus, duplicate detection algorithms developed to circumvent these difficulties. We display in Figure (3.23) a duplicate detection system.

Figure 3.23: Duplicate detection system.

The different phases (see Neiling et al. [2003]) are as follows:

1. The blocking phase (pre-selection): It tries to efficiently find a candidate set of duplicates. This is to save time since the comparison of object-pairs as well as the final matching decision are in general expensive. Therefore, the number of such comparisons must be heavily reduced before the detailed comparison can take place. A good candidate set of duplicates is a small set of pairs that contains all true duplicates. In order to find such a candidate set, the detection models must be evaluated with statistical techniques including recall and precision

[5] An object-pair is a vector of attribute-pairs.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(see Section (3.3.7)). The major problem is efficiency, that is, finding a good candidate set with a minimum of resources. In general, a blocking method should fulfil the following requirements:

- no false misses (very high recall) (it tells us how complete the results are)
- few false matches (high precision) (it tells us how valid the results are)
- few and cheap comparisons (low cost)
- little or no user-interaction (easy configuration)

Since the precision can be improved during the matching phase, one should focus on the recall.

2. The matching phase (comparison and classification): It is doing the actual duplicate decision. It is made of two sub-phases.

   (a) Comparison: It involves an in-depth comparison of the candidate duplicate pairs by comparing the individual attribute-pairs, which results in a comparison vector $\gamma[a, b]$ for every candidate pair. Here, the element γ_i represents the comparison results between the individual attribute-pairs. They can be boolean (attribute matches or does not match), discrete (such as matches, possibly matches, or does not match), or continuous (such as attribute values having a similarity of 0.2). The goal of a comparison algorithm is to estimate the degree of similarity between attribute values. Ideally an equivalent attribute value would always get a higher similarity score than a non-equivalent value. Thus, the requirements for a good comparison method are as follows:

   - clear separation (accurate similarity estimation)
   - little or no user-interaction (easy configuration)

   However, since the cause for differences between equivalent values are numerous, getting a perfect similarity function is a hard problem.

   (b) Decision models (Classification): The task for the matching algorithm is to classify such a given comparison vector γ into the sets M or U by some kind of decision model. The major problem is accuracy, that is, minimising the number of false matches and false misses. Further, the number of possible duplicates must be small. A decision model should ideally fulfil the following requirements:

   - high accuracy (high recall and high precision)
   - few possible matches (few clerical reviews)
   - little or no user-interaction (easy configuration)

   The higher the recall and precision for the automatically decided duplicate pairs, the higher will also be the number of unclear/possible duplicate pairs, and vice versa. The main problem in classification is to understand the relevance of the different attributes for the duplicate decision, that is, how to decide if one attribute matches, but another does not. In general, the individual attributes do not have the same relevance. For example, in the person domain a matching **person name** is a much stronger indication for a duplicate than a matching **year of birth**, because of the higher uniqueness of **person name** in contrast to **year of birth**. On the other hand, a non-matching **year of birth** can be a strong indication for a non-duplicate, when we assume a high reliability for **year of birth**. Another problem is when the individual values depend on each other, which is typically the case.

In what follows, we assume that the data is stored in tables having comparable fields or attributes. We also assume that we have identified which fields to compare (the blocking phase). We are left with matching duplicate records. We are going to first consider techniques for measuring the similarity of individual fields, and then, we will describe techniques for measuring the similarity of entire records.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.5.1.2 Overview

The duplicate record detection task is highly data-dependent and there is no technique dominating all others across all data sets. This makes the choice of the best model difficult. While one can focus on the quality of the comparison techniques and not on the efficiency of the duplicate detection process, a central issue is that of improving the speed of duplicate detection. We distinguish two main approaches for duplicate record detection:

1. Ad hoc techniques that work quickly on existing relational databases. These techniques do not rely on the existence of training data and emphasise efficiency over effectiveness.

2. More fundamental techniques that are based on probabilistic inference models, and deep neural networks.

While probabilistic methods outperform ad hoc techniques in terms of accuracy, the ad hoc techniques work much faster and can scale to databases with hundreds of thousands of records. One solution is to use techniques that rely both on machine learning and probabilistic inference.

When dealing with typographical variations (or errors), duplicate detection typically relies on string comparison techniques. Winkler [2006] and Elmagarmid et al. [2007] reviewed some of the techniques used for matching fields with string data in the duplicate record detection context. As there is a large number of errors or transformations that may occur in real-life data, a large number of field comparison metrics were devised. Multiple methods have been developed for this task, and each method works well for particular types of errors. We give an overview of these techniques below:

1. Character-based similarity metrics: They are specifically designed to handle typographical errors and rely on character edition operations, such as deletions, insertions, substitutions and sub-sequence comparisons. Some of the metrics include edit distance, affine gap distance, Smith-Waterman distance, Jaro distance metric, and Q gram distance.

2. Token-based similarity metrics (or vector-space methods): When typographical conventions lead to rearrangement of words (such as **John Smith** versus **Smith, John**) character-level metrics fail to capture the similarity of the entities. Token-based metrics try to compensate for this problem. Some of the metrics include atomic strings, WHIRL, Q-Grams with tf.idf.

3. Phonetic similarity metrics: While [1] and [2] focus on the string-based representation of the database records, strings may be phonetically similar even if they are not similar in a character or token level. The phonetic similarity metrics are trying to address such issues and match such strings. Some of the metrics include Soundex, New York State Identification and Intelligence System (NYSIIS), Oxford Name Compression Algorithm (ONCA), metaphone and double metaphone.

4. Numeric similarity metrics: In general, the numbers are treated as strings (and compared using the metrics described above) or simple range queries, which locate numbers with similar values.

In the case of toponym matching, one can compute string similarity metrics as above or use

- Vector-space methods: They transform strings into vector representations, over which similarity computations are performed.

- Hybrid approaches: It combines string similarity metrics with vector-space methods, in order to improve effectiveness when matching names composed of multiple tokens.

However, in most real-life situations, the records consist of multiple fields. In that case, we must rely on more flexible metrics that can accommodate multiple similarity comparisons. Some of the existing methods can be broadly divided into three categories:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1. The approaches that rely on training data to learn how to match the records. This category includes
   - Probabilistic approaches: We distinguish the Bayes decision rule for minimum error, the Bayes decision rule for minimum cost, Decision with a Reject Region.
   - Supervised machine learning techniques:
   - Semisupervised machine learning techniques:
2. The approaches that rely on domain knowledge or on generic distance metrics to match records. This category includes
   - Active-learning-based techniques:
   - Distance-based techniques:
   - Rule-based approaches:

   approaches that use declarative languages for matching and approaches that devise distance metrics appropriate for the duplicate detection task.
3. Unsupervised machine learning techniques:

We are going to review some of these techniques below.

As discussed above, standard approaches to typographical errors and toponym matching involve computing a similarity metric (or function) between the names that are to be matched, such as an edit distance, or a heuristic, such as the Jaro-Winkler metric, and then taking a decision based on a threshold over the similarity value. Thus, a similarity function requires the definition of a threshold value in order to decide whether two different data instances match. In this context, threshold definition is a central problem. While the performance differences for the individual similarity metrics are relatively small, carefully tuning the similarity threshold becomes critical for achieving good results. Testing several string similarity metrics over the toponym matching task, Santos et al. [2017] showed that the methods based on supervised learning for combining multiple similarity metrics (avoiding the manual tuning of the similarity threshold), such as ensembles of decision trees, could achieve good results on this task, significantly outperforming the individual similarity metrics.

With the explosion of data, data cleaning is becoming critical and we need techniques that focus both on the quality of the comparison and on the efficiency of the duplicate detection process. Alenazi et al. [2016] discussed issues related to scalability and efficiency for duplication detection techniques. He proposed eliminating the sorting process by using tree structure to improve the record duplication detection. Another approach is to consider adaptive methods that detect different patterns for duplicate detection and automatically adapt themselves over time. For example, a background process could monitor the current data, incoming data, and any data sources that need to be merged or matched, and decide, based on the observed errors, whether a revision of the duplicate detection process is necessary or not. We also need to develop methods that permit the user to derive the proportions of errors expected in data cleaning projects. We will need to improve information extraction techniques to enlarge the databases with internal and external information (unstructured text and from the web).

3.5.1.3 Using graph theory

As discussed in Section (3.2.1.2), a graph database stores the same sort of data as a relational database, but it is also capable of storing linkages between objects. Graph databases are closer to object-oriented structures than relational databases. Employing nodes, properties and edges, graph databases do not rely on join operations. Since the schema is less strict, it can handle fluctuating data changes quickly. Graph-like queries, such as figuring the shortest path through a series of nodes in a graph, are handled rapidly and naturally. For example, John buys a lot of Pepsi, Jack

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

is married to Valerie and buys different drinks. We do not have to run JOINs to understand how we should market to each individual customer. We can see the relationships in the data without having to make a hypothesis and test it. To understand this, linked lists / binary trees / heaps (anything that is linked) is faster than lists / tables / arrays. This is because if A is linked to B and you are already at A, then you do not have to search the memory for B. It is already there as it is connected to A. However, in ordinary lists and arrays you have to iterate through them entirely in order to reach B from A. Generally, if the data is about humans, there is some form of hierarchy or relations which are better depicted in graphs than tables. Graphs store both the data and the relations between the data. So, you can see relationships between data without having to do any extra work. This makes it easier to see similarities in the data for clustering.

Since most industries are now using graph databases rather than relational data bases, academics are no longer finding solutions for duplication issues on older types of databases whereas many advances are being made with regard to graph databases. A very efficient way for deduplication is clustering nodes based on similarities and then zooming into each cluster to find the duplicates. Clustering based on similarity is easy with graphs. Nowadays, Graph Theory (see Section (13.1)) has as different solution for each kind of duplication issue and more are being developed.

3.5.2 Ad hoc techniques

3.5.2.1 Character-based similarity metrics

We present a few character-based similarity metrics and refer the readers to Cohen et al. [2003], Winkler [2006] and Elmagarmid et al. [2007] for more details. Previous studies have shown that the performance of different approximate string similarity algorithms is task dependent and that there is no single overall best technique (see Cohen et al. [2003], Recchia et al. [2013]).

3.5.2.1.1 The edit distance Edit distance metrics are character-based methods that calculate the minimum number of edit operations, between two strings σ_1 and σ_2, needed to transform σ_1 into σ_2. The simplest edit distance, called the Levenshtein distance (see Levenshtein [1966]), between two strings σ_i, $i = 1, 2$, is the minimum number of edit operations of single characters needed to transform the string σ_1 into σ_2. There are three types of edit operations:

- insert a character into the string
- delete a character from the string
- replace one character with a different character

In general, each edit operation has cost 1. For example, the edit distance between the toponyms **Lisboa** and **Lisbonne** is three, corresponding to one substitution and two insertions, namely $a \to n$, $\epsilon \to n$, and $\epsilon \to e$ (see Santos et al. [2017]). The distance metric can be computed through a dynamic programming algorithm, and a corresponding normalised string similarity measure can defined as

$$s(\sigma_1, \sigma_2) = 1 - \frac{d(\sigma_1, \sigma_2)}{\max(|\sigma_1|, \sigma_2)}$$

where $d(\sigma_1, \sigma_2)$ corresponds to the Levenshtein edit distance between strings σ_1 and σ_2 with lengths $|\sigma_1|$ and $|\sigma_2|$, respectively. The dynamic programming algorithm for computing the edit distance between two strings takes $O(|\sigma_1| \cdot |\sigma_2|)$ time for two strings of length $|\sigma_1|$ and $|\sigma_2|$, respectively (see Navarro [2001]). Needleman et al. [1970] modified the original edit distance model and allowed for different costs for different edit distance operations. Ristad et al. [1998] presented a method for automatically determining such costs from a set of equivalent words that are written in different ways. However, the edit distance metrics work well for catching typographical errors, but they are typically ineffective for other types of mismatches.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.5.2.1.2 The affine gap distance Note, the edit distance metric does not work well when matching strings that have been truncated or shortened such as "John R. Smith" versus "Jonathan Richard Smith". As a solution, the affine gap distance introduced two extra edit operations: (1) open gap, (2) extend gap. Since the cost of extending the gap is usually smaller than the cost of opening a gap, it results in smaller cost penalties for gap mismatches than the equivalent cost under the edit distance metric. The algorithm for computing the distance requires $O(a \cdot |\sigma_1| \cdot |\sigma_2|)$ time when the maximum length of a gap is $a << \min(|\sigma_1|, |\sigma_2|)$. In the general case, the algorithm runs in approximately $O(a^2 \cdot |\sigma_1| \cdot |\sigma_2|)$.

3.5.2.1.3 The Jaro distance metric Jaro [1970] introduced a string comparison algorithm mainly used for comparison of last and first names. The basic algorithm for computing the Jaro metric for two strings σ_1 and σ_2 is as follows:

1. Compute the string lengths σ_1 and σ_2.
2. Find the common characters c in the two strings. Here, common are all the characters $\sigma_1[j]$ and $\sigma_2[j]$ for which $\sigma_1[i] = \sigma_2[j]$ and $|i - j| \leqslant \frac{1}{2}\min(|\sigma_1|, |\sigma_2|)$.
3. Find the number of transpositions t. The number of transpositions is computed as follows: We compare the ith common character in σ_1 with the ith common character in σ_2. Each nonmatching character is a transposition.

The Jaro comparison value is:

$$J(\sigma_1, \sigma_2) = \frac{1}{3}\Big(\frac{c}{|\sigma_1|} + \frac{c}{|\sigma_2|} + \frac{c - \frac{t}{2}}{2}\Big)$$

This algorithm algorithm requires $O(|\sigma_1| \cdot |\sigma_2|)$ time for two strings of length $|\sigma_1|$ and $|\sigma_2|$. This is mainly due to step [2], which computes the common characters in the two strings. Winkler et al. [1991] modified the Jaro metric to give higher weight to prefix matches since prefix matches are generally more important for surname matching.

3.5.2.1.4 Q-gram distance The q-grams are short character substrings of length q of the database strings (see Ukkonen [1992]). When two strings σ_1 and σ_2 are similar, they share a large number of q-grams in common. Given a string σ, its q-grams are obtained by sliding a window of length q over the characters of σ. Since q-grams at the beginning and the end of the string can have fewer than q characters from σ, the strings are conceptually extended by padding the beginning and the end of the string with $q - 1$ occurrences of a special padding character, not in the original alphabet. With the appropriate use of hash-based indexes, the average time required for computing the q-gram overlap between two strings σ_1 and σ_2 is $O(\max(|\sigma_1|, |\sigma_2|))$. The letter q-grams, including trigrams, bigrams, and/or unigrams, have been used in a variety of ways in text recognition and spelling correction (see Kukich [1992]). One natural extension of q-grams is the positional q-grams, which also record the position of the q-gram in the string (see Sutinen et al. [1995]). Gravano et al. [2001] showed how to use positional q-grams to efficiently locate similar strings within a relational database.

3.5.2.1.5 The Hamming distance The Hamming distance is mainly used for numerical fields of a fixed size, such as zip codes or social security numbers. It counts the mismatches between the two numbers being compared when determining the lowest number of substitutions needed (or the lowest number of errors incurred) when deriving one string from another.

3.5.2.2 Token-based similarity metrics

Token-based metrics try to account for typographical conventions that lead to rearrangement of words such as **John Smith** versus **Smith, John**. For instance, some vector-space approaches consist in computing the cosine similarity metric between representations based on character n-grams (based on sequences of n consecutive characters, typically

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

with $n = 2$ or $n = 3$). In practice, different variants of the cosine metric have been used such as leveraging binary representations for the occurrence of n-grams, versus using the common information retrieval term weighting heuristic known as Term Frequency times Inverse Document Frequency (TF-IDF). One can also consider the Dice similarity coefficients (see Dice [1945]), when computed between sets of character n-grams occurring in the strings to be compared. Further, rather than using regular n-grams, some studies have suggested that skip-grams, that is, bi-grams of non-adjacent letters, considering gaps of zero, one, or two characters, are also an effective choice for representing the strings to be matched (see Keskustalo et al. [2003]).

3.5.2.2.1 Atomic strings Monge et al. [1996] proposed a basic algorithm for matching text fields based on atomic strings, which are sequences of alphanumeric characters delimited by punctuation characters. Two atomic strings match if they are equal, or, if one is the prefix of the other. Based on this algorithm, the similarity of two fields is the number of their matching atomic strings divided by their average number of atomic strings.

3.5.2.2.2 WHIRL Cohen [1998] proposed WHIRL as a system that adopts from information retrieval the cosine similarity combined with the tf.idf weighting scheme to compute the similarity of two fields. One separates each string σ into words and each word w is assigned a weight

$$v_\sigma(w) = \log(tf_w + 1) \cdot \log(idf_w)$$

where tf_w is the number of times that w appears in the field, and idf_w is $\frac{|D|}{n_w}$, where n_w is the number of records in the database D that contain w. The tf.idf weight for a word w in a field is high if w appears a large number of times in the field (large tf_w) and w is a sufficiently rare term in the database (large idf_w). For example, for a collection of company names, relatively infrequent terms such as **AT&T** or **IBM** will have higher idf weights than more frequent terms such as **Inc.**. The cosine similarity of σ_1 and σ_1 is defined as

$$sim(\sigma_1, \sigma_2) = \frac{\sum_{j=1}^{|D|} v_{\sigma_1}(j) \cdot v_{\sigma_2}(j)}{\|v_{\sigma_1}\|_2 \cdot v_{\sigma_2}\|_2}$$

This metrics works well for a large variety of entries and is insensitive to the location of words, thus allowing natural word moves and swaps. Further, the introduction of frequent words only minimally affects the similarity of the two strings due to the low idf weight of the frequent words. However, it does not capture word spelling errors, especially if they are pervasive and affect many of the words in the strings. For example, the strings **Compter Science Department** and **Deprtment of Computer Scence** will have zero similarity under this metric. Bilenko et al. [2003] proposed the SoftTF-IDF metric to solve this problem, where pairs of tokens that are similar [6] (and not necessarily identical) are also considered in the computation of the cosine similarity. However, the product of the weights for non-identical token pairs is multiplied by the similarity of the token pair, which is less than one.

3.5.2.2.3 Q-Grams with tf.idf Gravano et al. [2003] extended the WHIRL system to handle spelling errors by using q-grams, instead of words, as tokens. In this setting, a spelling error minimally affects the set of common q-grams of two strings. For example, the two strings **Gteway Communications** and **Comunications Gateway** have high similarity under this metric, despite the block move and the spelling errors in both words. This metric handles the insertion and deletion of words nicely. The string **Gateway Communications** matches with high similarity the string **Communications Gateway International** since the q-grams of the word **International** appear often in the relation and have low weight.

3.5.2.2.4 Hybrids While hybrid metrics combine the advantages of character-based and vector-spaced approaches, most of these methods are based on applying a sub-measure to all pairs of word tokens between the two strings (they rely on second-level measures), and then computing a final score based on these values. For instance, Monge et al.

[6] The token similarity is measured using a metric that works well for short strings, such as edit distance and Jaro.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1996] proposed to compute the average similarity between the most similar pairs of word tokens, according to a sub-measure such as the Jaro-Winkler similarity. One can also use the cosine similarity or the Jaccard coefficient as hybrid approaches, by considering sets of tokens instead of sets of n-grams, and softening the metrics by allowing for small mismatches (by applying a threshold over the results of an inner similarity metric) when aligning the individual word tokens (see Moreau et al. [2008]).

3.5.3 Probabilistic matching models

Newcombe et al. [1959] first considered duplicate detection as a Bayesian inference problem. This approach was then formalised by Fellegi et al. [1969]. We let A and B denote the tables that we want to match with n comparable fields. In the duplicate detection problem, each tuple (object) pair $< \alpha, \beta >$ ($\alpha \in A$, $\beta \in B$) is assigned to one of the two classes M and U. The class M contains the record pairs that represent the same entity (match) and the class U contains the record pairs that represent two different entities (nonmatch or unmatch). Each tuple (object) pair $< \alpha, \beta >$ is expressed as a random vector (comparison vector) $x = (x_1, ..., x_n)^\top$ with n components corresponding to the n comparable fields of A and B. Each x_i shows the level of agreement of the ith field for the records α and β. In general, we use binary values for the element x_i, with $x_i = 1$ if the field i agrees and $x_i = 0$ if it disagrees. The comparison vector x is the input to a decision rule that assigns x to U or to M. It is assumed that x is a random vector whose density function is different for each of the two classes. Hence, if the density function for each class is known, the duplicate detection problem becomes a Bayesian inference problem. We want to determine whether $< \alpha, \beta > \in U$ or $< \alpha, \beta > \in M$.

3.5.3.0.1 Bayes decision rule for minimum error We write a simple probabilistic decision rule as

$$< \alpha, \beta > \in \begin{cases} M \text{ if } p(M|\, x) \geqslant p(U|\, x) \\ U \text{ otherwise} \end{cases}$$

where $p(M|\, x)$ is the probability of the match class M given the comparison vector x. Using Bayes theorem, this decision rule can be written as

$$< \alpha, \beta > \in \begin{cases} M \text{ if } l(x) = \frac{p(x|\, M)}{p(x|\, U)} \geqslant \frac{p(U)}{p(M)} \\ U \text{ otherwise} \end{cases}$$

where the ratio

$$l(x) = \frac{p(x|\, M)}{p(x|\, U)}$$

is the likelihood ratio. The probability distributions

$$m(x) = p(x|\ < \alpha, \beta > \in M) \text{ and } u(x) = p(x|\ < \alpha, \beta > \in U)$$

are the conditional probabilities of x, given α and β are elements on M and U, respectively. The ratio $\frac{p(U)}{p(M)}$ is the threshold value of the likelihood ratio for the decision. Belin et al. [1995] developed a method for setting that threshold. This rule is called the Bayes test for minimum error. When the conditional distributions $p(x|\, M)$ and $p(x|\, U)$, and the priors $P(U)$ and $P(M)$, are known, it results in the smallest probability of error and, as such, it is an optimal classifier (see Hastie et al. [2009]). However, this is rarely the case. In naive Bayes, one makes a conditional independence assumption and postulates that the probabilities $p(x_i|\, M)$ and $p(x_j|\, M)$ are independent if $i \neq j$ (we can replace M with U). In that case, the probability distributions can be written as

$$p(x|\, M) = \prod_{i=1}^{n} p(x_i|\, M) \text{ and } p(x|\, U) = \prod_{i=1}^{n} p(x_i|\, U)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The values of $m_i(x_i) = p(x_i|\, M)$ and $u_i(x_i) = p(x_i|\, U)$ can be computed using a training set of pre-labelled record pairs. The authors showed that these probabilities could be computed from the data under three restrictions: (1) x consists only of three components, (2) these components are conditionally independent, (3) the components are boolean (match, non-match). Note, the probabilistic model can also be used without using training data. In order to compute the probabilities m_i and u_i, $i = 1, .., n$, Jaro [1989] used a binary model for the values of x_i. That is, $x_i = 1$ if the field i matches, and $x_i = 0$ otherwise, and suggested using an expectation maximisation (EM) algorithm (see Dempster et al. [1977]) to compute the conditional probabilities $p(x_i = 1|\, M)$. The probabilities $p(x_i = 1|\, U)$ can be estimated by taking random pairs of records (which are with high probability in U). When the conditional independence is not a reasonable assumption, Winkler [1993] suggested using the general expectation maximisation algorithm to estimate the conditional distributions $p(x|\, M)$ and $p(x|\, U)$. Larsen et la. [2001] tried to explicitly model the dependencies between the probabilities by using clerical input to tune the model. Winkler [2002] claimed that the general, unsupervised EM algorithm works well under five conditions:

1. the data contain a relatively large percentage of matches (more than 5 percent),
2. the matching pairs are well-separated from the other classes,
3. the rate of typographical errors is low,
4. there are sufficiently many redundant identifiers to overcome errors in other fields of the record,
5. the estimates computed under the conditional independence assumption result in good classification performance.

The author showed that a semisupervised model, which combines labelled and unlabelled data performed better than purely unsupervised approaches. Du Bois [1969] proposed a different method to correct mismatches that occur due to missing values. He suggested using a new comparison vector x^* with dimension $2n$ instead of the n-dimensional comparison vector x such that

$$x^* = \big(x_1, ..., x_n, x_1 y_1, ..., x_n y_n\big)$$

where

$$y_i = \begin{cases} 1 \text{ if the ith field on both records is present} \\ 0 \text{ otherwise} \end{cases}$$

Using this representation, mismatches that occur due to missing data are typically discounted, resulting in improved duplicate detection performance. The author proposed using an independence model to learn the distributions of $p(x_i y_i|\, M)$ and $p(x_i y_i|\, U)$ by using a set of pre-labelled training record pairs. Ravikumar et al. [2004] considered the most general case for the parameter estimation of the model by using continuous valued attributes and taking dependencies into account. They used a hierarchical latent variable graphical model (HGM) to model the dependencies between the continuously valued individual attributes. They showed that this approach outperforms methods that discretise the continuous values into boolean values and methods that rely on the conditional independence assumption.

3.5.3.0.2 Bayes decision rule for minimum cost In practice, the misclassifications of M and U samples may have different consequences, so that the minimisation of the probability of error is not the best criterion for creating decision rules. One solution is to assign a cost c_{ij} to each situation, which is the cost of deciding that x belongs to the class i when actually belongs to the class j. The expected costs $r_M(x)$ and $r_U(x)$ of deciding that x belongs to the class M and U, respectively, are:

$$r_M(x) = c_{MM} \cdot p(M|\, x) + c_{MU} \cdot p(U|\, x) \text{ and } r_U(x) = c_{UM} \cdot p(M|\, x) + c_{UU} \cdot p(U|\, x)$$

In that case, the decision rule for assigning x to M becomes

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$< \alpha, \beta > \in \begin{cases} M \text{ if } r_M(x) < r_U(x) \\ U \text{ otherwise} \end{cases}$$

It was proved that the minimum cost decision rule for the problem can be stated as (see Duda et al. [1973]):

$$< \alpha, \beta > \in \begin{cases} M \text{ if } l(x) > \frac{(c_{MU} - c_{UU}) \cdot p(U)}{(c_{UM} - c_{MM}) \cdot p(M)} \\ U \text{ otherwise} \end{cases}$$

Note, when comparing the minimum error and minimum cost decision rule, we notice that the two decision rules become the same for the special setting of the cost functions to $c_{UM} - c_{MM} = c_{MU} - c_{UU}$. In this case, the cost functions are termed symmetrical. For a symmetrical cost function, the cost becomes the probability of error and the Bayes test for minimum cost specifically addresses and minimises this error.

3.5.3.0.3 Decision with a reject region Even though using the Bayes decision rule when the distribution parameters are known leads to optimal results, even in an ideal scenario, when the likelihood ratio $l(x)$ close to the threshold, the error (or cost) of any decision is high. Fellegi et al. [1969] suggested adding an extra reject class in addition to the classes M and U. The reject class contained record pairs for which it is not possible to make any definite inference and a clerical review is necessary. These pairs are examined manually by experts to decide whether they are true matches or not. By setting thresholds for the conditional error on M and U, we can define the reject region and the reject probability, which measure the probability of directing a record pair to an expert for review. Focusing on the costs of the decision, Tepping [1968] presented a graphical approach for estimating the likelihood thresholds. Verykios et al. [2003] developed a formal framework for the cost-based approach taken by Tepping which shows how to compute the thresholds for the three decision areas when the costs and the priors $p(M)$ and $p(U)$ are known. Verykios et al. [2004] extended the reject region approach to a larger number of decision areas. However, one problem with such a generalisation is appropriately ordering the thresholds which determine the regions in such a way that no region disappears.

3.5.4 Supervised and semisupervised learning

3.5.4.1 Overview

In general, these algorithms rely on similarity measures (or functions) which decides whether the data is the same or not by setting a threshold. In this context, threshold definition is a central problem. A lot of research developed around automatically estimating the optimum threshold. A lot of models developed that combine methods of similarity metrics with supervised learning techniques for proper weighing of record fields. More recently, the use of deep neural networks developed without the need of major tunings of the network parameters.

3.5.4.1.1 Supervised learning The supervised learning systems rely on the existence of training data in the form of record pairs, pre-labelled as matching or not. One approach is to treat each record pair $< \alpha, \beta >$ independently, similarly to the probabilistic techniques described above. For example, Cochinwala et al. [2001] used (1) the well-known CART algorithm (see Breiman et al. [1984]), which generates classification and regression trees, (2) a linear discriminant algorithm, which generates a linear combination of the parameters for separating the data according to their classes, and (3) a vector quantisation approach, which is a generalisation of nearest neighbour algorithms. They found that CART had the smallest error percentage. Elfeky et al. [2002] proposed to use machine learning techniques either based on supervised training of a classifier (some kind of decision model), or using unsupervised clustering methods like K-means. However, simple clustering methods like K-means are only able to identify clusters that are linear separable, which is in general not the case for real world data. The model, called TAILOR, was developed whose users can tune the system parameters and plug in the in-house developed and public domain tools to build their own record linkage models providing a framework for the record linkage process. Tejada et al. [2001] [2002] proposed a supervised learning method generating rules describing which combination of thresholds on the individual

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

attributes are necessary to declare a pair as duplicate. For example, if the similarity between the names is higher than a threshold T_1 and the similarity between the street name is higher than a threshold T_2, then declare it as duplicate. Bilenko et al. [2002] [2003] proposed a structure to improve the duplicate record detection using trainable measures of textual similarity. They considered a learnable text distance functions for each database field to illustrate that such measures are proficient enough to adapt to the precise notion of similarity suitable for the field's domain. They used the SVMlight to learn how to merge the matching results for the individual fields of the records (the learnable measure). The SVM model transforms input data points into a higher dimensional space, where they are then linear separable. The authors showed that the SVM approach usually outperformed simpler approaches, such as treating the whole record as one large field. Bilenko et al. [2005] extended the approach to an online learning scenario, where the data is streaming. Cohen et al. [2002] proposed a supervised approach in which the system learns from training data how to cluster together records that refer to the same real-world entry. That is, the model learns a hypothesis function that labels pairs as matching or unmatching. They proposed an adaptive distance function, which is learned from a given set of training examples. In the case where duplicates within one data source should be clustered, a confidence measure in the labelling for a match is used as edge weight in a graph that connects every potential duplicate object.

Sarawagi et al. [2002] addressed the problem of manually selecting a good training set for supervised learning approaches by presenting an active learning approach, which selects relevant pairs for manual labelling. They showed that this leads to a significantly reduced number of required labelled pairs. dos Santos et al. [2011] proposed a method to assess the quality of a similarity function at different threshold and choose an appropriate threshold for a specific application. The estimation process is based on a clustering phase performed over a data collection and requires no human intervention since the choice of similarity threshold is based on the silhouette coefficient (internal quality measure for clusters). Isele et al. [2011] used genetic programming (GP) (see Section (19.3)) to automatically generates linkage rules from a set of reference links. Similarly, de Carvalho [2013] used genetic programming to automatically propose duplicate record detection function by combining several pieces of evidence taken from the data. This function is able to identify whether the two records in a repository are same or not. In order to define the threshold in applications where the similarity functions are used, Deepa et al. [2013] found the optimal equation using Genetic Algorithm (GA), and then adopted an intelligence algorithm, Particle Swarm Optimisation (PSO) to get the optimal threshold to detect the duplicate records more accurately. Similarly, Ranganathana et al. [2018] focused on improving the accuracy of the duplicate record detection process, along with other well-known measures, by adopting different optimisation algorithms and getting the optimal threshold automatically to find the duplicate records more accurately minimising human intervention. To do so, they adopted Adaptive Neuro Fuzzy Inference Systems (ANFIS) (see Yazdi et al. [2010]) for duplicate record detection by means of similarity functions, and they used Genetic Algorithm (GA) and Particle Swarm Optimisation (PSO) for generating the optimal similarity measure which decides whether the data is the same or not.

Testing several string similarity metrics over the toponym matching task, Santos et al. [2017] showed that the methods based on supervised learning for combining multiple similarity metrics (avoiding the manual tuning of the similarity threshold), such as ensembles of decision trees, could achieve good results on this task, significantly outperforming the individual similarity metrics. However, these techniques still struggled handling cases that involve complex transliterations and highly dissimilar strings (such as involving different alphabets). Santos et al. [2018] proposed a string matching approach using deep neural networks to classify pairs of toponyms as either matching or non-matching. They used Gated Recurrent Units (GRUs), a type of recurrent neural network architecture that can be used for modelling sequential data (see Section (6.2)), to build representations from the sequences of bytes that correspond to the strings that are to be matched. These representations are then combined and passed to feedforward nodes, finally leading to a classification decision. Here, the network takes toponym pairs (two sequences of bytes) as input, and outputs a binary classification decision. They showed that their model could significantly outperform individual string similarity metrics used in previous studies, as well as previous methods based on supervised machine learning for combining multiple similarity metrics. Extending on this approach, de Sousa Marinho [2018] used deep neural networks for string matching by considering architectures integrating shortcut connections, self attention mechanisms, hard alignment attention between the strings being compared, or max-pooling over the sequences of outputs of the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

recurrent layers. The author also considered a concatenation of the representations of the strings obtained through neural network layers, with other additional features. These features are derived from the dataset attributes but can also be related to the similarity between the strings. These models achieved superior results on all datasets, when compared to string similarity measures, without the need of major tunings of the network parameters.

3.5.4.1.2 Semisupervised learning Pasula et al. [2002] examined the problem of citation matching, which consist in identifying if a given citation refers to a specific paper or not. They proposed a semisupervised probabilistic relational model that can handle a generic set of transformations. Even though the model can handle a large number of duplicate detection problems, the use of exact inference resulted in a computationally intractable model. The authors proposed using a Markov Chain Monte Carlo (MCMC) sampling algorithm as approximation method to avoid the intractability issue. However, it is unclear whether techniques that rely on graph-based probabilistic inference can scale well for data sets with hundreds of thousands of records. de Freitas et al. [2010] presented the Active Learning GP (AGP), a semi-supervised GP for the data deduplication problem, that uses an active learning approach in which a committee of multi-attribute functions votes for categorising record pairs as duplicates or not. When the voting is not enough to predict the class of the data pairs, then a user is called in order to resolve the conflict. Results showed that AGP guarantees the quality of the deduplication.

3.5.4.2 Examples

3.5.4.2.1 Finding the optimal threshold Deepa et al. [2013] found the optimal equation using Genetic Algorithm (GA), and then adopted an intelligence algorithm, Particle Swarm Optimisation (PSO) to get the optimal threshold to detect the duplicate records more accurately. This was extended by Ranganathana et al. [2018] who proposed the ANFIS (see Yazdi et al. [2010]) for duplicate record detection by means of similarity functions, which are used to find the similarity between records and fields.
We combine both approaches which we present below. The steps of the framework are as follows:

1. Similarity computation: Similarity functions compute the similarity of each field with other record field and assign a similarity value for each field. Thus, one must choose an appropriate similarity function (or measure) for each field.

2. Generate feature vector: Each element of the feature vector represents the similarity function applied on the values of a specific attribute, for example $< name, levenshtein >$. Here, the approach combines several similarity values obtained from different similarity measures to compute the distance between any two records. That is, one develop a set of expressions by using these vectors elements with the simple mathematical operators $(+, -, \times, /)$ (see Figure (3.25)).

3. Optimised expression: A set of such expression are supplied as input to GA to find best among the supplied inputs which is capable of providing better solution for the problem. The optimisation algorithm PSO find the optimal threshold for each expression.

The workflow of the system is displayed in Figure (3.24).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 3.24: Duplicate ANFIS workflow.

The workflow of the system is described as follows:

1. Training phase: The dataset contains the labelled match and unmatch which are trained separately using ANFIS. First, an object-pair is considered and the similarity is computed using several similarity measures to obtain the similarity distance between two individual records. Here, the similarity computation was carried out for all pairs of records using Levenshtein distance and Cosine similarity measures. Then, these similarity values are combined to generate feature vectors. Both the steps are repeated for all the pair of records in the training dataset to obtain a set of feature vectors. Finally, we train the ANFIS with these vectors.

2. Duplicate detection phase: A simple clustering method such as K-means is used to identify clusters in the testing dataset. The comparison is done for each of the data records present only within the cluster to reduce the number of record comparisons. Pairs of records that fall under each cluster are the candidates for a full similarity computation. Then the record comparison is performed as in step [1], creating the distance feature vectors. Finally, these vectors are fed to ANFIS as input to obtain a binary matrix that distinguishes the duplicate and the non-duplicate records based on the threshold value.

One can consider either a gradient descent or hybrid learning rule to optimise the system locally. In [1], the similarity values obtained from different similarity measures are combined to generate feature vectors. The authors used Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) (see Section (19.1)) to extract these features. The population is initialised with the generated set of expressions. This is illustrated in Figure (3.25), where a, b, c, d corresponds to similarity values defined on the attributes name, address, phone number and category, respectively.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(a+b)+(c+d)

(a+b)*(c+d)

(a-b)+(c+d)

(a+b)*(c-d)

(a+b)-(c-d)

Figure 3.25: initial populations provided by the set of expressions.

The accuracy is calculated for each such expression. New populations are generated by applying mutation and crossover for finding the fittest expressions among the other expressions in the population. The fitness value is a value generated from the fitness function. Here, the authors have used the $F1$ metric as the fitness function (see Section (3.3.7)). It is calculated as follows:

- Precision (P): It is defined as the fraction of identified duplicate pairs that are correct.

$$P = \frac{\text{Number of true duplicate records identified}}{\text{Total number of duplicate records identified}}$$

- Recall (R): It is the fraction of actual duplicate pairs that are identified correctly in the input dataset.

$$R = \frac{\text{Number of true duplicate records identified}}{\text{Total number of duplicate records present in the dataset}}$$

  Recall can be seen as a measure of completeness, whereas precision is a measure of exactness or fidelity.

- $F1$-measure (F): It gives equal weight to both precision and recall and it is the harmonic mean of precision and recall. The traditional F-measure or balanced F-score is computed as

$$F = \frac{2 \times R \times P}{R + P}$$

We want to find the fitness value for each expression in the population based on threshold value. Since the $F1$-value varies with different threshold, it is necessary to choose an optimised threshold to classify the dataset as duplicates and non-duplicates accurately. Hence, the authors applied one of the PSO to find the global optimal solution. In PSO, a population starts with a random set of threshold (particle) on each expression. The position of the particle refers to the possible solutions to be optimised. Next, we find the fitness value for each such particle using $F1$-metric and determine particle best and global best (see Section (19.1.2.2)). The particles move towards the optimal area by updating their position and the velocity. PSO select the best threshold value on each such expression which classify the set of records as duplicate and non-duplicate. Then, Select the best n expressions having high fitness value and apply crossover and mutation to generate new set of population. Repeat the process until termination criteria is reached. Eventually, the best expression with a high fitness value is selected as the best solution for determining the duplicates. Once the optimal similarity measure is obtained from the training data, the duplicate detection in testing dataset is carried out using the same measure. This is illustrated in Figure (3.26).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 3.26: Extracting the feature vectors.

Recall, while the optimal solution obtained after ending criteria may be the best that can be produced, it is not always the best solution for the problem. To remedy this, the authors have considered the exploration and exploitation of several scenarios. The former deals with the convergence of the unlikely solutions to a single group and finding the best from it, while the latter deals with exploring the best solution from the likely solutions. They proposed a modified scheme of PSO and ABC to detect the duplicate records. The notation for the algorithm is

- ν^0 is current velocity
- ν_{ex} is updated velocity
- p_{best} is current best position
- pos^0 is current position
- g_{best} is global best position
- φ, ϕ random values in range $[0, 1]$

The pseudocode for the modified PSO algorithm is given in the Algorithm (1).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 1 Modified PSO algorithm

Require: Input:
Require: Create random population
Require: Initialise population
1: **repeat**
2: Generate random p_{best} values

$$P = \left[p_1, p_2, ..., p_n\right]$$

3: Calculate fitness
4: Calculate corresponding p_{best}

$$p_i \rightarrow [+, *, ...] :: \left[p_{i-1}, p_{i+1}, ..., p_n\right] \rightarrow P_i^{corresponds}$$

5: Find g_{best}

$$P_i^{corresponds} \Rightarrow p_i \leftarrow g_{best}$$

6: Apply crossover / mutation

$$g_{best} \Leftarrow \text{crossover / mutation}$$

7: Generate new populations

$$\begin{aligned} \nu_{ex} &= \nu^0 + \varphi(p_{best} - pos^0) + \phi(g_{best} - pos^0) \\ pos &= pos^0 + \nu_{ex} \end{aligned}$$

8: Calculate fitness
9: **if** $new_{fitness} > old_{fitness}$ **then**
10: replace g_{best}
11: **end if**
12: **until** stopping criteria

The performance is evaluated under various thresholds on different datasets.

3.5.4.2.2 Using NLP The work Santos et al. [2018] and de Sousa Marinho [2018] showed that recurrent neural networks (RNNs) could achieve better results at detecting duplicates on pairs of string when compared to traditional string similarity measures. They explored different architectures, specially the ones related to the Natural Language Inference task, on the string matching task. The main idea being to use sentence pair modelling from natural language processing (NLP) (see Section (14.5)) to identify duplicates. The neural architectures from this NLP problem can easily be adapted to address the string matching problem, considering strings as input sequences of characters, instead of sequences of words (see Lan et al. [2018]). This is illustrated in Figure (3.27).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 3.27: Typical string NLP model.

Since Gated Recurrent Units (GRUs) can be used to model sequential data or for generating new sequences, Santos et al. [2018] used GRUs within a deep model that considers multiple layers of recurrent and feed-forward processing. The NN architecture for addressing the toponym matching problem is illustrated in Figure (3.28).

Figure 3.28: NN architecture for toponym matching.

The steps of this architecture are as follows:

- Representation: The input to the network are two sequences of binary vectors that represent the strings to be compared. To create these vectors, the strings are first converted to a unicode canonical normalised format (see Section (3.1.3.2)) (using a fully decomposed UTF-8 representation, in which all combining character marks are placed in a pre-specified order) and they are also padded [7] with a special symbol denoting the beginning and termination of the toponym. The normalised strings are then represented as a sequence of one-hot binary vectors (see Section (3.1.3.3)), in which a single bit is set to one for each byte in the sequence that corresponds to the unicode normalised string.

[7] In cryptography, padding is any of a number of distinct practices which all include adding data to the beginning, middle, or end of a message prior to encryption.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Vectorisation: The binary vectors are input to bi-directional GRUs, producing a vector of real values, also referred to as an embedding, for each of the toponyms being compared. The bi-directional GRUs work by concatenating the outputs of two GRUs, one processing the sequence from left to right and the other from right to left. Here, the architecture uses two different layers of bi-directional recurrent units:

  1. The first bd-layer generates a sequence of real-valued vectors, that is then passed as input to the second bi-directional GRU layer.
  2. The second bd-layer outputs a single embedding for the input, resulting from the concatenation of the last outputs that are produced by the GRUs that process the input sequences in each direction.

- Combining the representations: The authors considered siamese networks (see Mueller et al. [2016]). In these networks, while each of the input toponyms is conceptually processed by individual recurrent layers, in order to produce a compact representation for its contents, the parameters of these GRUs are shared across the parts of the network that process each input toponym. Thus, the two layers of bi-directional GRUs used to process each of the input strings have the same set of parameters. The two embeddings produced by the bi-directional GRU layers must then be combined and compared. Following Mou et al. [2016], the authors produced a combined representation for the pair of toponyms, via simple operators $(+, -, \times, /)$ (matching methods), resulting from

  1. the concatenation of the embedding vectors
  2. the element-wise product of the embedding vectors
  3. the difference between the embedding vectors

  The results from each matching method are concatenated, resulting in the final representation that is going to be used for the classification.

- Feed-forward network: This combined representation is then passed as input to feed-forward network layers:

  1. the first layer uses a simple combination of the inputs together with a non-linear activation function (such as a rectified linear unit).
  2. the second layer produces the final output with a sigmoid activation function.

The entire network is trained with back-propagation in combination with the Adam optimisation algorithm, using binary cross-entropy as the loss function to be optimised. To control overfitting and improve the generalisation, the authors used dropout regularisation with a probability of 0.1 between each layer of the proposed neural network architecture.

3.5.4.3 Graph-based approach

Graph-based decision models build on the idea that taking relationships between different objects into account increases the overall duplicate detection accuracy. They build up graph structures between the objects and then either try to propagate information about inferred duplicates through the graph or find some global optimal solution. While the graph-based decision models are very promising approaches, they suffer from the complexity problem of building up such a graph-based data structure and the computational problem of calculating a globally optimal solution.
We now introduce supervised clustering techniques that have records as nodes for the graph: Suppose that have a set of n documents to cluster into topics, but do not know the topics. However, given two documents A and B, we have a classifier $f(A, B)$ that outputs whether or not it believes A and B are similar to each other. Here, f was learned from some past training data. One approach to clustering is to apply f to every pair of documents in the set, and then to find the clustering that agrees as much as possible with the results. This is solved with graph theory: we have a complete graph on n vertices (items), where each edge (u, v) is labelled either $+$ or $-$ depending on whether u and v have been deemed to be similar or different. We want a clustering that maximises (agreement) the number of $+$ edges

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

within clusters, plus the number of $-$ edges between clusters (equivalently, minimises the number of disagreements: the number of $-$ edges inside clusters plus the number of $+$ edges between clusters). This formulation is motivated from a document clustering problem in which one has a pairwise similarity function f learned from past data, and the goal is to partition the current set of documents in a way that correlates with f as much as possible.

A typical post-processing step for the above techniques (including the probabilistic techniques) is to construct a graph for all the records in the database, linking together the matching records. Then, using the transitivity assumption, all the records that belong to the same connected component are considered identical (see Monge et al. [1997]). However, the transitivity assumption can sometimes result in inconsistent decisions. For example, the pairs $< \alpha, \beta >$ and $< \alpha, \gamma >$ can be considered matches, but not the pair $< \beta, \gamma >$. Partitioning such inconsistent graphs with the goal of minimising inconsistencies is an NP-complete problem. Bansal et al. [2004] proposed a polynomial approximation algorithm that can partition such a graph, automatically identifying the clusters and the number of clusters in the data set. They gave a constant factor approximation to the problem of minimising disagreements, and a PTAS (polynomial-time approximation scheme) for maximizing agreements.
Cohen et al. [2000b] proposed to find an optimal global solution and proved that this task was NP-hard, and instead propose a priority-queue based algorithm. Similarly to Cohen et al. [2002], McCallum et al. [2004] learned the clustering method using training data. They replaced the generative model of Pasula et al. [2002] with conditional models called conditional random fields. Similarly to the work of Bansal et al. [2004], their technique is equivalent to a graph partitioning technique that tries to find the min-cut and the appropriate number of clusters for the given data set. They showed the effectiveness of this approach for the task of noun coreference for natural language processing, and also for the task of citation matching.
Singla et al. [2004] [2005] considered the problem of similar entries that can not be detected by string distance metrics. They build up a graph containing all candidate pairs using conditional random fields. Pairs that are clear duplicates, but contain different values in individual attributes, are used to infer that these values are equivalent. This information is then propagated through the graph and used for further duplicate detection. However, instead of using records as nodes for the graph, they proposed to use attribute values as nodes, to propagate information across nodes and improve duplicate record detection. For example, if the records

$$< \text{Google}, \text{MountainView}, \text{CA} > \text{ and } < \text{Google Inc.}, \text{MountainView}, \text{California} >$$

are deemed equal, then **CA** and **California** are also equal and this information can be useful for other record comparisons. The underlying assumption is that the only differences are due to different representations of the same entity (such as **Google** and **Google Inc.**) and that there is no erroneous information in the attribute values (for example, by mistake someone entering Bismarck,ND as the location of Google headquarters). Following a similar approach, Dong et al. [2005] built up a graph and propagate duplicate information to related objects.

3.5.5 Unsupervised learning

In supervised learning, the comparison space consists of comparison vectors which contain information about the differences between fields in a pair of records. Unless some information exists about which comparison vectors correspond to which category (match, nonmatch, or possible match), the labelling of the comparison vectors in the training data set should be done manually. One way around is to use clustering algorithms and group together similar comparison vectors. The idea behind most unsupervised learning approaches for duplicate detection is that similar comparison vectors correspond to the same class. It has its roots in the probabilistic model proposed by Fellegi et al. [1969]. Recall, when there is no training data to compute the probability estimates, it is possible to use variations of the Expectation Maximisation (EM) algorithm to identify appropriate clusters in the data.
Verykios et al. [2000] proposed to use a bootstrapping technique based on clustering to learn matching models. It uses very few labelled data, and then use unsupervised learning techniques to appropriately label the data with unknown labels. They treated each entry of the comparison vector (which corresponds to the result of a field comparison) as a continuous, real variable. Then, they partition the comparison space into clusters by using the AutoClass clustering

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

tool (see Cheeseman et al. [1996]). Each cluster contains comparison vectors with similar characteristics, so that all the record pairs in the cluster belong to the same class (matches, nonmatches, or possible matches). Hence, by knowing the real class of only a few vectors in each cluster, it is possible to infer the class of all vectors in the cluster and, therefore, mark the corresponding record pairs as matches or not. The authors showed that the classifiers generated using the new, larger training set have high accuracy, and require only a minimal number of pre-labelled record pairs. Following a similar approach, Ravikumar et al. [2004] proposed a hierarchical, graphical model for learning to match record pairs. It is based on modelling each field of the comparison vector as a latent binary variable which shows whether the two fields match or not. The latent variable then defines two probability distributions for the values of the corresponding observed comparison variable. They showed that was easier to learn the parameters of a hierarchical model than to attempt to directly model the distributions of the real valued comparison vectors. Bhattacharya [2005] proposed to use the Latent Dirichlet Allocation generative model to perform duplicate detection. In this model, the latent variable is a unique identifier for each entity in the database. Lehti [2006] proposed a way to automatically select a representative sample of non-duplicates that can be used by the decision models to increase their decision accuracy. This would allow to detect duplicates not only on flat records, but also on inter-related objects. He also extended the Fellegi-Sunter model for record linkage for determining the parameters without the independence assumption and using continuous distance values. He proposed a probabilistic model that simplifies the problem of finding a good threshold on the result, the handling of null values, multi-valued attributes and relationships

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 4

Ensemble models

In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. A machine learning ensemble consists of only a concrete finite set of alternative models, but typically allows for much more flexible structure to exist among those alternatives.
We briefly introduce ensemble models. We are first going to describe some well known models for classification and regression such as boosting, bagging and decision trees. We will then introduce the concepts of bias and variance for classifiers to better understand stability and instability of these algorithms. We will then use convex potential functions to formalise the approach, and we will briefly describe random forests.
More details can be found in textbooks by Rokach [2010], Zhou [2012], among others.

4.1 The base models

4.1.1 Introduction

4.1.1.1 Presentation

Supervised learning algorithms perform the task of searching through a hypothesis space to find a suitable hypothesis that will make good predictions with a particular problem (see Section (10.1)).However, even if the hypothesis space contains hypotheses that are very well-suited for a particular problem, it may be very difficult to find a good hypothesis. Ensemble models combine multiple hypotheses to form a (hopefully) better hypothesis. The term ensemble is usually reserved for methods that generate multiple hypotheses using the same base learner.
Ensemble methods train multiple learners to solve the same problem by combining them. It contains a number of learners called **base learners** generated from a base learning algorithm (decision tree, neural network etc). The base learners are also referred to as weak learners. A **weak learner** is just slightly better than random guess, while a **strong learner** is very close to perfect performance [1].
Early contributions to ensemble methods are:

- combining classifiers (for pattern recognition working on strong classifiers),
- ensembles of weak learners (machine learning boosting performance from weak to strong, such as, AdaBoost, Bagging, etc), and

[1] For any distribution, with high probability, given polynomially many examples and polynomial time, we can find a classifier with generalisation error better than random guessing.

$$\epsilon < \frac{1}{2}\ , \text{ also denoted } \gamma > 0 \text{ for generalisation error } (\frac{1}{2} - \gamma)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- mixture of experts (neural networks considering a divide-and-conquer strategy). It consider a divide-and-conquer strategy, try to learn a mixture of parametric models jointly and use combining rules to get an overall solution.

The two pieces of pioneering work are:

- Hansen et al. [1990] found that predictions made by the combination of a set of classifiers are often more accurate than the ones made by the best single classifier.
- Schapire [1990] proved that weak learners (easy to obtain) can be boosted to strong learners.

Generally, an ensemble is constructed in two steps: generating the base learners, and then combining them.
Stacking (also called stacked generalisation), proposed by Wolpert [1992], is a popular method for improving results on data mining competitions. It is a model ensembling technique used to combine information from multiple predictive models to generate a new model (see Figure (4.1)). It is assumed that the stacked model (also called 2nd-level model) will outperform each of the individual models due to its smoothing nature and ability to highlight each base model where it performs best and discredit the ones where it poorly performs. Thus, stacking is most effective when the base models are significantly different. Some of the advantages of the stacking techniques are to avoid over-fitting due to K fold cross validation and to interpret non-linearity between features due to treating output scores as features.

4.1.1.2 Definitions

We briefly introduce ensemble methods and then present a few facts on model stacking.

Definition 4.1.1 *An ensemble is a set of classifiers that learn a target function, and their individual predictions are combined to classify new examples.*

An ensemble is constructed in two steps, generating the base learners, and then combining them. There are two basic ways of aggregating classification methods:

1. Before the fact: Creates solutions to be combined (for example bagging).
   Suppose we want to create a robust method for determining if a given image contains a face:
   - Problem:It may be very difficult and computationally intensive to create a classifier for that task
   - One solution: detect eyes instead of faces
     - Advantage: A lot more efficient
     - Problem: this may produce models with low accuracy
     - Solution: Combine it with other similar classifiers
2. After the fact: Combines existing solutions (for example blending).
   Netflix challenge teams merging: Different models are already built. Find an intelligent way of combining them. Given each hypothesis $\widehat{f}_1, ..., \widehat{f}_M$ and a new instance x, we can use a linear regression to compute the prediction $g(x)$. That is,

$$\widehat{f}_1, ..., \widehat{f}_M \to g(x) = \sum_{i=1}^{M} \omega_i \widehat{f}_i(x)$$

   Choose ω_i to minimise the error on aggregation set.

Each of these methods can be more or less appropriate to particular problems. There are two paradigms of ensemble methods:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1. Sequential ensemble methods: the base learners are generated sequentially (AdaBoost). It exploits the dependence between the base learners, since the overall performance can be boosted in a residual-decreasing way.

2. Parallel ensemble methods: the base learners are generated in parallel (Bagging). It exploits the independence between the base learners, since the error can be reduced dramatically by combining independent base learners.

Some examples of ensemble methods are bagging, boosting, random forests, extra trees. Nowadays, stacking and blending are also ensemble methods (see details in textbook by Zhou [2012]). We now briefly introduce stacking:

Definition 4.1.2 *Model stacking is a learning method that aims to improve predictive accuracy by combining predictions from multiple models. It is a particular case of ensemble learning.*

It is argued that model stacking leads to better predictions, at the expense of interpretability [2]. In general, it is used as a component of several winning entries in Kaggle competitions.

Figure 4.1: Two levels model stacking

4.1.1.3 An optimisation problem

An ensemble is itself a supervised learning algorithm, because it can be trained and then used to make predictions. To do so, we let $\mathcal{X}$ and $\mathcal{Y}$ be the input and output spaces, $\mathcal{D}$ the probability distribution, D the data set (sample), H the set of hypotheses, $I(\cdot)$ the indicator function. We let x be an instance or feature vector. For example, we consider the training data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$ where the instances x_i are independent and identically distributed (i.i.d.) and $y_i = f(x_i)$ where f is the ground-truth target function. That is, x_i is a vector and y_i takes value in $\mathbb{R}$.
In supervised learning, the learning process aim at constructing a learner h minimising the generalisation error (see Section (10.1.2)). Some examples of loss function are:

- In classification:

$$\mathcal{E}(h) = E_x[I(h(x) \neq f(x))]$$

  For instance, we let $\mathcal{X}$ be the instance space, $\mathcal{Y}$ be the label space, define the labelled data set as $L = \{(x_1, y_1), ..., (x_l, y_l)\}$ and consider the binary classification tasks where $\mathcal{Y} = \{-1, +1\}$.

- In regression: we want to minimise the squared error

$$\mathcal{E}(h) = E_x[\big(h(x) - f(x)\big)^2]$$

[2] In mathematical logic, interpretability is a relation between formal theories that expresses the possibility of interpreting or translating one into the other.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In general, trained ensemble models represent a single hypothesis which is not necessarily contained within the hypothesis space of the models from which they are built. Thus, ensembles can be shown to have more flexibility in the functions they can represent.

4.1.1.4 Some characteristics

4.1.1.4.1 Diversity We call ensemble diversity the difference among the individual learners. The diversity is crucial to ensemble performance since a gain from combination requires the individual learners to be different. The major problem is that the individual learners are trained for the same task from the same training data, and thus they are usually highly correlated. However, several algorithms assume either independent or less correlated learners. Even if correlation between individual learners is considered, it is generally based on the assumption that the individual learners produce independent estimates of the posterior probabilities, which is not the case in practice. In addition, the problem of generating diverse individual learners is even more challenging if we consider that the individual learners must not be very poor, and otherwise their combination would not improve and could even worsen the performance. For example, when the performance of individual classifiers is quite poor, the added error of simple soft voted ensemble may become arbitrarily large. As a result, it is desired that the individual learners should be accurate and diverse. Nonetheless, combining only accurate learners is often worse than combining some accurate ones together with some relatively weak ones, since complementarity is more important than pure accuracy. In conclusion, the success of ensemble learning lies in achieving a good tradeoff between the individual performance and diversity. Unfortunately, there is no well-accepted formal definition of diversity, but it is crucial in ensemble learning.

4.1.1.4.2 Generalisation error Ensemble methods are based on statistical tools and, as such, rely on error computation and error reduction. For instance, in parallel ensemble methods (bagging), the error can be reduced dramatically by combining independent base learners. As an example, we consider binary classification on classes $\{-1, +1\}$ and let f be the truth function. We suppose that each base classifier has an independent generalisation error ϵ, that is, for base classifier h_i

$$P(h_i(x) \neq f(x)) = \epsilon$$

After combining N number of such base classifiers according to

$$H(x) = \text{sgn}\Big(\sum_{i=1}^{N} h_i(x)\Big)$$

the ensemble H (prediction) makes an error only when at least half of its base classifiers make errors. By Hoeffding inequality, the generalisation error of the ensemble is

$$P(H(x) \neq f(x)) = \sum_{k=0}^{\lfloor \frac{N}{2} \rfloor} \binom{N}{k} (1-\epsilon)^k \epsilon^{N-k} \leqslant e^{-\frac{1}{2}N(2\epsilon-1)^2}$$

It shows that the generalisation error reduces exponentially to the ensemble size N and ultimately approaches to zero as N approaches to infinity. Even though one can not get absolutely independent base learners generated from the same training data set, base learners with less dependence can be obtained by introducing randomness in the learning process, and a good generalisation ability can be expected by the ensemble. Further, parallel methods are favourable to parallel computing, speeding up the training process.
One can show that the generalisation error of an ensemble depends on a term related to diversity. Thus, one must estimate error decomposition to understand ensemble learning. Examples are the error-ambiguity decomposition and the bias-variance decomposition.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.1.1.4.3 Perturbing and combining Some classification and regression methods are unstable in the sense that small perturbations in their training sets or in construction may result in large changes in the constructed predictor. Unstable methods can have their accuracy improved by perturbing and combining (PnC) (see Freund et al. [1995]). That is, by generating multiple versions of the predictor by perturbing the training set or construction method and then combining these versions into a single predictor. For instance, Ali [1995] generates multiple classification trees by choosing randomly from among the best splits at a node and combines trees using maximum likelihood. Breiman [1996e] adds noise to the response variable in regression to generate multiple subset regressions and then averages these.

4.1.2 Some classical models

4.1.2.1 The k-nearest neighbour

The k-nearest neighbour (kNN) algorithm relies on the principle that objects similar in the input space are also similar in the output space. It does not have an explicit training process, but simply stores the training set instead. As such, it is a lazy learning approach. For a test instance, a k-nearest neighbour learner identifies the k instances from the training set that are closest to the test instance. Then, for classification, the test instance will be classified to the majority class among the k instances. On the other hand, for regression, the test instance will be assigned the average value of the k instances. Figure (4.2) illustrates how to classify an instance by a 3-nearest neighbour classifier.

Figure 4.2: A k-nearest neighbour classifier

4.1.2.2 Binary space partitioning

Binary space partitioning (BSP) is a method for recursively subdividing a space into convex sets by hyperplanes. This subdivision gives rise to a representation of objects within the space by means of a tree data structure known as a BSP tree. It was developed in the context of 3D computer graphics (see Schumacker et al. [1969], Fuchs et al. [1980]), where the structure of a BSP tree allows spatial information about the objects in a scene that is useful in rendering, such as their ordering from front-to-back with respect to a viewer at a given location, to be accessed rapidly. Other applications include performing geometrical operations with shapes (constructive solid geometry) in CAD, collision

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

detection in robotics and 3D video games, ray tracing and other computer applications that involve handling of complex spatial scenes.
BSP is a generic process of recursively dividing a scene into two until the partitioning satisfies one or more requirements. The specific choice of partitioning plane and criterion for terminating the partitioning process varies depending on the purpose of the BSP tree. It arose from the computer graphics need to rapidly draw three-dimensional scenes composed of polygons. One disadvantage with BSP is that generating a tree can be time-consuming. Typically, it is therefore performed once on static geometry, as a pre-calculation step, prior to rendering or other real time operations on a scene. The expense of constructing a BSP tree makes it difficult and inefficient to directly implement moving objects into a tree.
For simplicity of exposition we describe the partitioning of a 2D space. Any 2D space X can be partitioned with a line L_1 into two subspaces X_1 and X_2 as illustrated in Figure (4.3).

Figure 4.3: Binary space partitioning with line L_1

We can further partition that space with the line L_{1-1} obtaining the subspace X_{1-1} and X_{1-2}. This partitioning can be represented with a tree in Figure (4.4).

Figure 4.4: Binary space partitioning with line L_1 and L_{1-1}

It leads to further partitioning of the subspace X_{1-1} into X_{1-1_1} and X_{1-1_2}. This partitioning can be represented with a new branching as illustrated in Figure (4.5).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 4.5: Binary space partitioning with line L_1 and L_{1-1} with partitioning tree

Adding another line, we can keep refining the subspaces and go down the tree as illustrated in Figure Figure (4.6).

Figure 4.6: Binary space partitioning with line L_1, L_{1-1} and L_{1-1-1} with partitioning tree

This process can be repeated so that the space X is represented with a binary partitioning tree (BPT). Any arbitrary partitioning of any arbitrary 2D space can be represented with a BPT.

4.1.2.3 Decision tree

Decision tree (DT) learning is one of the most popular techniques for regression and classification. The regression or classification model learnt through these techniques is represented as a tree called a decision tree. It consists in a set of tree-structured decision tests working in a divide-and-conquer way as follows:

- Each non-leaf node is associated with a feature test also called a split, which splits data into different subsets according to their different values on the feature test.
- Each leaf node is associated with a label, which will be assigned to instances falling into this node.

As an example of decision tree, we consider the parse tree, the regression case, and the classification case.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.1.2.3.1 Parse tree A parse tree is graph-theoretical structure representing single algebraic expressions (programs). For example, the tree t in Figure (4.7) represents the function $t(x) = x \times (1 + x) = x + x^2$.

Figure 4.7: A decision tree for regression

The vertices of the parse tree are classified as

1. root: top vertex
2. terminal vertices: leaf vertices. They correspond either to the independent variable(s) $x, y, ..$ or to some constants $0, 1, 2, ...$
3. function vertices: all other vertices (and top vertex). They correspond to simple operations $unaryf(a), binaryf(a, b), ternary$

For prescribed set of function and terminal vertices we may define a set of all possible parse trees that are composed of the prescribed type vertices $T = \{t_i; i = 1, ..., N\}$. Each parse tree $t \in T$ is identified with its interpretation function $t(x)$. Thus, the set T may be understood as a set of functions.

4.1.2.3.2 Regression When predicting with a tree, a series of feature tests is conducted, starting from the root node, and the result is obtained when a leaf node is reached. We consider a given a training set of points (regression table)

$$A_{train} = \{(x_i, y_i); i = 1, ..., N\}$$

The goal of standard regression analysis (see Section (10.1)) is to find optimal parameter(s) of a given function, $h(x, W)$, where W is a weight matrix, so that the following objective function (with any norm) is minimised

$$\begin{aligned} E(W) &= \sum_{i=1}^{N} |h(x_i; W) - y_i| \\ W^* &= \arg\min_{W} E(W) \end{aligned}$$

Thus, the adapted function $h(x; W^*)$ models the training set (regression table).
Going one step further, symbolic regression (SR) analysis (see Section (19.3.6.1)) looks for a function in a set (space) T so that the following objective function is minimised

$$\begin{aligned} E(t) &= \sum_{i=1}^{N} |t(x_i) - y_i| \\ t^* &= \arg\min_{t \in T} E(t) \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In genetic programming (GP), these functions can be represented by parse trees (composed of prescribed function and terminal vertices).

4.1.2.3.3 Classification On the other hand, the classification process starts by testing whether the value of the feature y-coordinate is larger than a threshold. For example, 0.73 in Figure (4.8). If so, the instance is classified as "cross", and otherwise the tree tests whether the feature value of x-coordinate is larger than another threshold (here 0.64); if so, the instance is classified as "cross" and otherwise is classified as "circle".

Figure 4.8: A decision tree in classification

4.1.2.3.4 Selecting the splits The process is recursive since in each step, a data set is given and a split is selected, then this split is used to divide the data set into subsets, and each subset is considered as the given data set for the next step. It remains to decide how to select the splits.
For instance, dividing the training set D into subsets $D_1, ..., D_k$, Breiman et al. [1984] proposed the CART which uses Gini index for selecting the split maximising the Gini

$$G_{gini}(D; D_1, ..., D_k) = I(D) - \sum_{i=1}^{k} \frac{|D_k|}{|D|} I(D_k)$$

where

$$I(D) = 1 - \sum_{y \in \mathcal{Y}} P^2(y|D)$$

and $P(y|D)$ is the conditional probability of y given the set D.
Alternatively, Quinlan [1998] proposed to use the information gain criterion in the ID3 algorithm. Given a training set D, the entropy of D (see Definition (2.2.1)) is defined as

$$\text{Ent}(D) = -\sum_{y \in \mathcal{Y}} P(y|D) \log P(y|D)$$

If the training set D is divided into subsets $D_1, ..., D_k$, the entropy may be reduced. The amount of the reduction is the information gain

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$G(D; D_1, ..., D_k) = \text{Ent}(D) - \sum_{i=1}^{k} \frac{|D_k|}{|D|} \text{Ent}(D_k)$$

Thus, the feature-value pair which will cause the largest information gain is selected for the split. However, features with a lot of possible values will be favoured, disregarding their relevance to classification. Note, Quinlan [1993] had considered a variant of the information gain called the $C4.5$ based on the gain ratio

$$P(D; D_1, ..., D_k) = G(D; D_1, ..., D_k) \cdot \Big(\sum_{i=1}^{k} \frac{|D_k|}{|D|} \log \frac{|D_k|}{|D|} \Big)^{-1}$$

which takes normalisation on the number of feature values. In that setting, the feature with the highest gain ratio, among features with better-than-average information gains, is selected as the split. Note, $C4.5$ and CART can deal with numerical features.

To reduce the risk of overfitting, a general strategy is to employ pruning to cut off some tree branches caused by noise or peculiarities of the training set. Pre-pruning tries to prune branches when the tree is being grown, while post-pruning re-examines fully grown trees to decide which branches should be removed. This is achieved with the validation error: for pre-pruning, a branch will not be grown if the validation error will increase by growing the branch; for post-pruning, a branch will be removed if the removal will decrease the validation error.

Trees that are grown very deep tend to learn highly irregular patterns, but they overfit their training sets. That is, they have low bias, but very high variance.

4.1.2.4 Boosting

Boosting is an algorithm that reduces bias and variance by converting weak learners to strong learners (see Section (4.1.1.4.2)). Multiple classifiers are constructed and vote for classes. Here, **the construction is sequential**, with the construction of the $(k+1)$st classifier depending on the performance of the k previously constructed classifiers. These algorithms are called Adaptive Resampling and Combining, or Arcing algorithms. The main idea being that the points most likely to be selected for the replicate data sets are those most likely to be missclassified. Focusing on these troublesome points may do better than other methods.

4.1.2.4.1 The procedure We suppose the weak learner will work on any given data distribution, and consider the binary classification task as an example. That is, we want to classify instances as positive and negative. The training instances in space $\mathcal{X}$ are drawn i.i.d. from distribution $\mathcal{D}$, and the truth function is denoted by f. Suppose the space $\mathcal{X}$ is made of three parts $\mathcal{X}_i$, $i = 1, 2, 3$, each taking $\frac{1}{3}$ amount of the distribution, and a learner working by random guess has 50% classification error on this problem. We want to get an accurate classifier on that problem, but can only have a weak classifier with correct classifications in both spaces $\mathcal{X}_1$ and $\mathcal{X}_2$, and a wrong classifications in space $\mathcal{X}_3$. Thus, we have $\frac{1}{3}$ classification error. We denote this weak classifier as h_1. The idea of boosting is to correct the mistakes made by classier h_1 by deriving a new distribution $\mathcal{D}^{'}$ from $\mathcal{D}$, which makes the mistakes of classifier h_1 more evident. For example, it focuses more on the instances in space $\mathcal{X}_3$. Then, we can train a classifier h_2 from the distribution $\mathcal{D}^{'}$. Again, suppose h_2 is also a weak classifier, which has correct classifications in both spaces $\mathcal{X}_1$ and $\mathcal{X}_3$, and a wrong classifications in space $\mathcal{X}_2$. By combining the classifiers h_1 and h_2 in an appropriate way, the combined classifier will have correct classifications in space $\mathcal{X}_1$, and maybe some errors in spaces $\mathcal{X}_2$ and $\mathcal{X}_3$. Again, we derive a new distribution $\mathcal{D}^{''}$ to make the mistakes of the combined classifier more evident, and train a classifier h_3 from the distribution, so that h_3 has correct classifications in spaces $\mathcal{X}_2$ and $\mathcal{X}_3$. Then, by combining the classifiers h_1, h_2 and h_3, we have a perfect classifier, since in each space of $\mathcal{X}_i$, $i = 1, 2, 3$, at least two classifiers make correct classifications. Generalising this example, boosting works by training a set of learners sequentially and combining them for prediction, where the later learners focus more on the mistakes of the earlier learners. See implementation details in Algorithm (2).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 2 Boosting procedure

Require: Input: sample distribution $\mathcal{D}$, base learning algorithm L, number of learning rounds N
Require: $\mathcal{D}_1 = \mathcal{D}$ initialise distribution
1: **for** $n = 1$ **to** N **do**
2: $h_n = L(\mathcal{D}_n)$ train a weak learner from distribution $\mathcal{D}_n$
3: $\epsilon_n = P_{x \sim \mathcal{D}_n}(h_n(x) \neq f(x))$ evaluate the error of h_n
4: $\mathcal{D}_{n+1} =$ Adjust-Distribution $(\mathcal{D}_n, \epsilon_n)$
5: **end for**
Require: Output: $H(x) =$ Combine-Outputs $(\{h_1(x), ..., h_n(x)\})$

This algorithm is designed to drive the training set error rapidly to zero.

4.1.2.4.2 Some algorithms While Algorithm (2) provides a technical description of a boosting algorithm, we still need to generate distribution and combine the outputs. The main variation between many boosting algorithms is their method of weighting training data points and hypotheses. For instance, Freund et al. [1995] [1997] proposed the AdaBoost to specify Adjust-Distribution and Combine-Outputs . There are many more recent algorithms such as LPBoost, TotalBoost, BrownBoost, xgboost, MadaBoost, LogitBoost, and others. Many boosting algorithms fit into the AnyBoost framework (Mason et al. [2000]), which shows that boosting performs gradient descent in a function space using a convex cost function.

We assume a probability distribution $\{p(n); n = 1, ..., N\}$ on the cases in the training set. Start with probability $p(n) = \frac{1}{m}$ and resample from distribution D to form the first training set $D^{(1)}$. We want to make sure that misclassified input data gain a higher weight, while examples that are classified correctly lose weight. Thus, future weak learners focus more on the examples that previous weak learners misclassified. Practically, as the sequence of classifiers and training sets is being built, increase probabilities $p(n)$ for those cases that have been most frequently missclassifed. A factor, $\beta > 1$, is defined which depends on the missclassification rate: the smaller it is, the larger is β. If the nth case in distribution D is missclassified by the classifier $C_{D^{(1)}}(x)$, then set the weight $\beta p(n)$ on that case, otherwise the weight is $p(n)$. Then, divide each weight by the sum of the weights to get the updated probabilities for the next round of sampling. At termination, combine classifiers by weighted (arc-fs) or, simple (unweighted arc-x4), voting. This is illustrated in Figure (4.9).

Figure 4.9: Bootsting with parallel learners and weighted dataset.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, the probability $p(n)$ is sometimes denoted $\mathcal{D}_n$
The construction of the distribution is as follows: We initialise the weight distribution $\mathcal{D}_1 = \frac{1}{m}$. Given $\mathcal{D}_n$ and h_n, we get

$$\mathcal{D}_{n+1} = \frac{\mathcal{D}_n}{Z_n} c(x)$$

where

$$c(x) = \begin{cases} e^{-\alpha_n} \text{ if } y = h_n(x) \\ e^{\alpha_n} \text{ if } y \neq h_n(x) \end{cases}$$

Z_n is a normalisation factor enabling $\mathcal{D}_{n+1}$ to be a distribution, and

$$\alpha_n = \frac{1}{2} \ln \beta_n > 0 \, , \, \beta_n = \frac{1 - \epsilon_n}{\epsilon_n}$$

where

$$\epsilon_n = P_{x \sim \mathcal{D}_n}(h_n(x) \neq f(x)) = \sum_k p(k) I(h_k(x) \neq f(x))$$

That is

$$\mathcal{D}_{n+1} = \frac{\mathcal{D}_n}{Z_n} e^{-\alpha_n y h_n(x)}$$

The combination of the output is given by

$$H(x) = \text{sgn}\Big(\sum_{n=1}^{N} \alpha_n h_n(x)\Big)$$

See implementation details in Algorithm (3).

Algorithm 3 AdaBoost algorithm

Require: Input: data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$, base learning algorithm L, number of learning rounds N
Require: $\mathcal{D}_1 = \frac{1}{m}$ initialise the weight distribution
1: **for** $n = 1$ **to** N **do**
2: $h_n = L(D, \mathcal{D}_n)$ train a classifier h_n from D under distribution $\mathcal{D}_n$
3: $\epsilon_n = P_{x \sim \mathcal{D}_n}(h_n(x) \neq f(x))$ evaluate the error of h_n
4: **if** $\epsilon_n > \frac{1}{2}$ **then**
5: break
6: **end if**
7: $\alpha_n = \frac{1}{2} \ln(\frac{1-\epsilon_n}{\epsilon_n})$ determine the weight of h_n
8: $\mathcal{D}_{n+1}(x) = \frac{\mathcal{D}_n(x)}{Z_n} e^{-\alpha_n f(x) h_n(x)}$ update the distribution, where Z_n is a normalisation factor which enables $\mathcal{D}_{n+1}$ to be a distribution
9: **end for**
Require: Output: $H(x) = \text{sgn}\big(\sum_{n=1}^{N} \alpha_n h_n(x)\big)$

AdaBoost can be seen as the algorithm B_ϕ corresponding to the potential function $\phi(x) = e^{-x}$. The training error is given by

$$H(x) \leqslant e^{-2\gamma^2 N}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

if for all n, we have $\gamma_n \geqslant \gamma > 0$. However, this training error only measures correctness of classifications, but neglects confidence of classifications. The latter can be measured as follows:

$$H(x) = \text{sgn}(f(x))$$

where $f(x) = \frac{\sum_n \alpha_n h_n}{\sum_n \alpha_n} \in [-1, 1]$. We let the margin be given by

$$\text{margin}(x, y) = yf(x)$$

Then we get

- $\text{margin}(x, y)$ close to $+1$ is high confidence, correct.
- $\text{margin}(x, y)$ close to -1 is high confidence, incorrect.
- $\text{margin}(x, y)$ close to 0 is low confidence.

Considering a binary classification on classes $\{-1, +1\}$, Friedman et al. [2000] modified the AdaBoost algorithm by minimising the exponential loss function

$$L_{exp}(h|\mathcal{D}) = E_{x\sim\mathcal{D}}[e^{-f(x)h(x)}]$$

using additive weighted combination of weak learners as

$$H(x) = \sum_{n=1}^{N} \alpha_n h_n(x)$$

Minimising the exponential loss by H, the partial derivative for every x is zero, that is, $\partial_{H(x)} e^{-f(x)H(x)} = 0$, so that

$$H(x) = \frac{1}{2} \ln \frac{P(f(x) = 1|x)}{P(f(x) = -1|x)}$$

As a result,

$$\text{sgn}(H(x)) = \arg \max_{y\in\{-1,1\}} P(f(x) = y|x)$$

which implies that $\text{sgn}(H(x))$ achieves the Bayes error rate. The output H is obtained by iteratively generating the classifier h_n and weight α_n.

4.1.2.4.3 Non-convex algorithms AdaBoost and LogitBoost are convex algorithms that can be defeated by random noise. As a result, they cannot learn basic and learnable combinations of weak hypotheses (see Long et al. [2007]). However, multiple authors demonstrated that boosting algorithms based on non-convex optimisation, such as BrownBoost, could learn from noisy datasets. Specifically, they can learn the underlying classifier of the Long-Servedio dataset. BrownBoost is an adaptive version of the boost by majority algorithm (see Freund [2001]).

4.1.2.5 Bagging

4.1.2.5.1 Description In statistics, bootstrapping is a metric that uses random sampling with replacement. It assigns measures of accuracy, such as bias, variance, confidence intervals, prediction error, to sample estimates. The simplest implementation of the idea of generating quasi-replicate training sets is bootstrap aggregating, also called bagging. Here, **the construction is spatial** via data perturbation. Bagging is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. In classification, multiple classifiers are constructed and vote for classes. In regression, bagging is a special case of the model averaging approach.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Bootstrap AGGregatING (called Bagging) is composed of bootstrap and aggregation (see Breiman [1996c]). Since the combination of independent base learners will lead to a dramatic decrease of errors, we want to get the base learners as independent as possible. Ideally, given a data set, we want to sample a number of non-overlapped data subsets and then train a base learner from each of the subsets. However, this is in general not possible due to limited data. One solution is to consider data sample manipulation, such as data perturbation. Here, bagging perturbs the training set repeatedly to generate multiple predictors and combines these by simple voting (classification) or averaging (regression).

4.1.2.5.2 The procedure Bagging adopts the bootstrap distribution (see Efron et al. [1993]) for generating different base learners. That is, it applies bootstrap sampling to obtain the data subsets for training the base learners. Given a training dataset $D = \{(x_1, y_1), ..., (x_m, y_m)\}$ containing m number of training examples, a sample of $m^{'}$ training examples will be generated by sampling with replacement. Repeating this process N times, bagging generates N new training sets D_n, by sampling from D uniformly and with replacement. If $m^{'} = m$, then, for large m the set D_n is expected to have the fraction $1 - \frac{1}{e} \approx 63.2\%$ of the unique examples of D, the rest being duplicates (see Aslam et al. [2007]).

Practically, set equal probabilities $p(n) = \frac{1}{m}$, $n = 1, ..., N$, on each case, and using these probabilities, sample with replacement (bootstrap) N times from the training set D, forming the resampled training set $D^{(B)}$. Note, some cases in D may not appear in $D^{(B)}$, and some may appear more than once. $D^{(B)}$ is called a bootstrap sample from D. The distribution on D given by the probabilities $\{p(n); n = 1, ..., N\}$ is denoted as $\mathcal{D}_{bs}$ (or $P^{(B)}$). Then, the sample $D^{(B)}$ is i.i.d. from $\mathcal{D}_{bs}$. By applying the process N times, N (independent) samples of m training examples are obtained. Then, from each sample D_n a base learner h_n can be trained by applying the base learning algorithm. The outputs of the base learners are aggregated by voting for classification and averaging for regression. This is illustrated in Figure (4.10).

Figure 4.10: Bootstrap aggregating.

If the base learners are able to output confidence values, weighted voting or weighted averaging are often used. For instance, when forecasting instance (feature vector), the algorithm feeds the instance to its base classifiers and collects all of their outputs, and then votes the labels and takes the winner label as the prediction, where ties are broken arbitrarily.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.1.2.5.3 Algorithm for classification For classification, use a training set D, an inducer L (or learning algorithm), and the number of bootstrap samples N as input. Generate a classifier H as output. The algorithm is as follows:

1. Create N new training sets D_n from D with replacement.
2. Classifier h_n is built from each set D_n using L to determine the classification of set D_n.
3. Finally classifier H is generated by using the previously created set of classifiers h_n on the original data set D. The classification predicted most often by the sub-classifiers h_n is the final classification. Mathematically, this is an $\arg\max$ problem for the label y over the output space $\mathcal{Y}$.

See details in Algorithm (4).

Algorithm 4 Bagging procedure

Require: Input: data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$, base learning algorithm L, number of base learner N

for $n = 1$ **to** N **do**

  $h_n = L(D, \mathcal{D}_{bs})$ where $\mathcal{D}_{bs}$ is the bootstrap distribution

end for

Require: Output: $H(x) = \arg\max_{y\in\mathcal{Y}} \sum_{n=1}^{N} I(h_n(x) = y)$

Note, bagging has a large variance reduction effect which is particularly effective with unstable base learners.

4.1.3 Bias and variance

Breiman [1996d] introduced the concepts of bias and variance for classifiers to better understand stability and instability of the the algorithms described above. Unstable classifiers can have low bias but high variance. The main effect of combining multiple versions, such as bagging and Arcing, is to reduce variance. In the latter, variance reduction comes from the adaptive resampling.

4.1.3.1 Regression

We present the bias and variance which we relate to the decomposition of prediction error. Given a training set $D = \{(x_n, y_n); n = 1, ..., N\}$ where the y_n are numerical outputs and the x_n are multidimensional input vectors, we assume that some method (neural nets, regression trees, etc) is applied to the data set to construct a predictor $f_D(x)$ of future y-values. Further, we assume that the training set D is made of i.i.d. samples from the distribution of X, Y and that future samples will be drawn from the same distribution. The squared error of f is given by

$$E_S(f_D) = E_{X,Y}[\big(Y - f_D(X)\big)^2]$$

We let $E_S(f)$ be the expectation of $E_S(f_D)$ over D. We can always decompose Y as

$$Y = f^*(X) + \epsilon$$

where $E[\epsilon|X] = 0$. We let $f_A(x) = E_D[f_D(X)]$ and define the bias and variance as

$$\begin{aligned} Bias(f) &= E_X[\big(f^*(X) - f_A(X)\big)^2] \\ Var(f) &= E_{D,X}[\big(f_D(X) - f_A(X)\big)^2] \end{aligned}$$

where $f_A(\cdot)$ is the aggregated predictor. Then, we get the fundamental decomposition

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$E_S(f) = E[\epsilon^2] + Bias(f) + Var(f)$$

It states that at each point x, the contribution to the error at x from the bias is $\big(f^*(x) - f_A(x)\big)^2$ and that from the variance is $E_D[\big(f_D(x) - f_A(x)\big)^2]$. In general, at each point x, both the contributions of the bias and that of the variance are positive.
Using this decomposition we can better understand the properties of the predictors. In general, some family $\mathscr{F}$ of functions is defined and f is selected as the function in $\mathscr{F}$ having minimum squared error over the training set. Thus, if $\mathscr{F}$ is the set of linear functions, and f^* is fairly non-linear, then the bias will be large. Further, if that set is small, estimating a small number of of parameters, the variance will be low. However, if $\mathscr{F}$ is a large family of functions, such as a large neural net or binary decision trees, then the bias is usually small, but the variance is large.
One solution for improving the bias is to enlarge the size of the family $\mathscr{F}$, but we increase the variance. To reduce the the variance we consider the aggregated predictor $f_A(x)$, which has the same bias than $f(x)$ but has zero variance. By approximating $f_A(x)$, we get a predictor with reduced variance.

4.1.3.2 Classification

In classification, the output variable $y \in \{1, ..., J\}$ is a class label, the training set is given by $D = \{(x_n, y_n); n = 1, ..., N\}$. Some method is used to construct a classifier $C_D(x)$ for predicting future y-values. We assume that the data in the training set is made of i.i.d. selections from the distribution of X, Y. The misclassification error is given by

$$E_M(C_D) = E_{X,Y}[C_D(X) \neq Y]$$

We denote $E_M(C)$ the expectation of $E_M(C_D)$ over D. Further, we denote the transition probability and the density as

$$P(j|x) = P(Y = j|\ X = x) \text{ and } P(dx) = P(X \in dx)$$

The minimum missclassification rate is given by the Bayes classifier C^* where

$$C^*(x) = \arg\max_j P(j|x)$$

with missclassification rate

$$E_M(C^*) = 1 - \int \max_j (P(j|x))P(dx)$$

We need to express the aggregated predictor in a discrete way. We let

$$Q(j|x) = P_D(C_D(x) = j)$$

and define the aggregated classifier as

$$C_A(x) = \arg\max_j Q(j|x)$$

which corresponds to aggregation by voting. Given many independent replicas $D_1, D_2, ...$, we construct the classifiers $C_{D_1}(x), C_{D_2}(x), ...$ and at each x we determine the classification $C_A(x)$ by having these multiple classifiers vote for the most popular class.

Definition 4.1.3 *$C_D(x)$ is unbiased at x if $C_A(x) = C^*(x)$.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

It states that $C_D(x)$ is unbiased at x if, over the replication of D, $C_D(x)$ selects the right class more often than any other class. However, it does not mean that it is as accurate classifier.
If C is unbiased at x, then $C_A(x)$ is optimal. Let U be the set of all x at which C is unbiased. The complement of U is called the bias set and denoted by B.

Definition 4.1.4 *The bias of a classifier C is*

$$Bias(C) = P_{X,Y}(C^*(X) = Y \ , \ X \in B) - E_D[P_{X,Y}(C_D(X) = Y \ , \ X \in B)]$$

and its variance is

$$Var(C) = P_{X,Y}(C^*(X) = Y \ , \ X \in U) - E_D[P_{X,Y}(C_D(X) = Y \ , \ X \in U)]$$

It leads to the fundamental decomposition

$$E_M(C) = E_M(C^*) + Bias(C) + Var(C)$$

Even though aggregating a classifier and replacing C with C_A reduces the variance to zero, there is no guarantee that it will reduce the bias. Actually, there are some cases where the bias will be increased. Thus, if the bias set B has large probability, $E_M(C_A)$ may be significantly larger than $E_M(C)$.
The bias and variance defined above have the following properties:

1. Bias and variance are always non-negative.
2. The variance of C_A is zero.
3. If C is deterministic (does not depend on D), then its variance is zero.
4. The bias of C^* is zero.

Proof: The variance of C can be expressed as

$$Var(C) = \int_U \big[\max_j P(j|x) - \sum_j Q(j|x)P(j|x)\big]P(dx)$$

Similarly, the bias of C is given by that integral with U replaced by B. So, we see that both bias and variance are non-negative. Since $C_A = C^*$ on U, its variance is zero. If C is deterministic, then $C = C^*$ on U, and C has zero variance. At last, C^* has no bias.
Note, in classification, each point x is either in the bias set or in the variance set. If it is in the bias set, then the variance at x is zero, and vice versa. It reflects the difference between classification and regression. In the former you either get it right or wrong, in the latter the error is continuous.

4.1.3.3 Instability

A prediction method is unstable in the sense that small changes in the training set can cause large changes in the resulting predictors. More formally, as D changes, the classifiers $C_D(x)$ can considerably differ from each other and from the aggregated classifier $C_A(x)$. On the other hand, stable classifiers do not change much over replicates of D, so that $C_D(x)$ and $C_A(x)$ will tend to be the same with small variance.
Breiman [1996c] [1996d] showed that trees, neural nets were unstable with high variance. Trees (such as CART) have high variance, but they are largely unbiased. If their variance can be reduced, they we may obtain accurate classifiers. Alternatively, nearest neighbours, linear discriminant analysis (LDA) are stable. LDA get stability by having a limited set of models to fit the data, but it can result in large bias if the data can not adequately be represented with that set of models.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In the case of bagging, $C_A(x)$ depends on the underlying probability P that the training sets are drawn from, that is, $C_A(x) = C_A(x, P)$. The bagged classifier is $C_A(x, P^{(B)})$. To reduce variance we want $C_A(x, P^{(B)})$ to be a good enough approximation to $C_A(x, P)$. In the case of Arcing (boosting), multiple classifiers are sequentially constructed and vote for classes. In general, Arcing does better than bagging at variance reduction. When increasing the number of classifiers, the error rates for Arcing decreases significantly towards the Bayes minimums. Bagging error rates do not decrease markedly.

4.1.4 Noise in random classification

The classical models described in Section (4.1.2) are supervised learning processes minimising a loss function. This is formalised in Section (10.1.1). The boosting and bagging algorithms presented in Section (4.1.2) belongs to the class of convex potential function (CPF) and are highly sensitive to random classification noise (see Long et al. [2007]). This is not the case with branching program based boosters which are not part of the CPF framework. We briefly present this result.

4.1.4.1 Convex potential functions

We let ϕ be a potential function and define convex potential functions as follows:

Definition 4.1.5 *We say that $\phi : \mathbb{R} \to \mathbb{R}$ is a convex potential function if ϕ satisfies the following properties:*

1. *ϕ is convex and nonincreasing and $\phi \in C^1$.*
2. *$\phi^{'}(0) < 0$ and $\lim_{x\to\infty} \phi(x) = 0$*

We denote B_ϕ the algorithm corresponding to the potential convex function ϕ.

4.1.4.1.1 Boosters A booster combines weak classifiers into a comprehensive prediction by using an optimally weighted majority vote of weak classifiers. It has been shown that boosting algorithms can be viewed as greedy iterative algorithms performing a coordinate-wise gradient descent to minimise a potential function of the margin of the examples (see Bradley et al. [2007]).
Given the instance space $\mathcal{X}$ we let $\mathcal{H} = \{h_1, ..., h_N\}$ denote a fixed finite collection of base classifiers over $\mathcal{X}$, where each base classifier is a function $h_i : \mathcal{X} \to [-1, 1]$. We let $S = (x_1, y_1), ..., (x_m, y_m) \in \big(\mathcal{X} \times \{-1, 1\}\big)^m$ denotes a multiset (see Appendix (23.2.1.6)) of m examples with binary labels. We consider the case where boosters are run on a distribution $\mathcal{D}_{\nu,S}$ obtained by starting with a finite multiset of examples, and adding independent misclassification noise. We can extend the definition of a booster to apply to probability distribution over $\mathcal{X} \times \{-1, 1\}$ by extending Definition (4.1.5) as follows

$$P_{\phi,\mathcal{D}}(\alpha_1, ..., \alpha_N) = E_{(x,y)\sim\mathcal{D}}[\phi\big(yF(x; \alpha_1, ..., \alpha_N)\big)]$$

where $F(x; \cdot)$ is the quantity whose sign is the outcome of the vote, and whose magnitude reflects the closness of the vote. It is given by

$$F(x; \alpha_1, ..., \alpha_N) = \sum_{i=1}^{N} \alpha_i h_i(x)$$

For rational values of ν, running the algorithm B_ϕ on the above probability for $\mathcal{D} = \mathcal{D}_{\nu,S}$ is equivalent to running B_ϕ over a finite multiset where each element of S occurs a number of times proportional to its weight under $\mathcal{D}$.
Some examples of convex potential functions are:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Convex potential boosters: all boosting algorithms choosing voting weights α_i, $i = 1, ..., N$, with output classifier

$$H(x) = \text{sgn}\big(\sum_{i=1}^{N} \alpha_i h_i(x)\big)$$

obtained by taking the resulting vote over the base classifier prediction.

- Global-minimising convex potential boosters: the algorithms choosing voting weights α_i, $i = 1, ..., N$, to minimise the global potential function over S:

$$B_\phi^{ideal} = P_{\phi,S}(\alpha_1, ..., \alpha_N) = \sum_{i=1}^{m} \phi\big(y_i F(x_i; \alpha_1, ..., \alpha_N)\big)$$

It is a convex function from $\mathbb{R}^N$ to $\mathbb{R}$.

- L_1-regularised boosters: For any $C > 0$ the L_1-regularised booster minimises $P_{\phi,S}$ subject to the constraint that $\sum_{i=1}^{N} |\alpha_i| \leqslant C$. It is denoted as $B_{\phi,C}^{L_1}$ (see Ratsch et al. [2001]).

4.1.4.2 A formalism

4.1.4.2.1 Boostable We fix a classifier $c : \mathcal{X} \to \{-1, 1\}$ and a multiset $S = (x_1, y_1), ..., (x_m, y_m)$ of examples labelled according to c. We say that a set of base classifiers $\mathcal{H} = \{h_1, ..., h_N\}$ is boostable with respect to c and S if there is a vector $\alpha \in \mathbb{R}^N$ such that for all $i = 1, ..., m$ we have

$$\text{sgn}\big(\alpha_1 h_1(x_i) + \cdots + \alpha_N h_N(x_i)\big) = y_i$$

If $\gamma > 0$ is such that

$$\frac{y_i \cdot \big(\alpha_1 h_1(x_i) + \cdots + \alpha_N h_N(x_i)\big)}{|\alpha_1| + \cdots + |\alpha_N|} \geqslant \gamma$$

for all i, we say that $\mathcal{H}$ is boostable with respect to c and S with margin γ. If it is the case, then a range of different boosting algorithms (such as AdaBoost) can be run on the noise-free data set S to efficiently construct a final classifier that correctly labels every example in S.

4.1.4.2.2 Noise tolerance Freund et al. [1996], Dietterich [2000] observed that AdaBoost could suffer poor performance when run on noisy data. The most commonly given explanation for this is that the exponential reweighting of examples which it performs (a consequence of the exponential potential function) can cause the algorithm to invest too much effort on correctly classifying noisy examples. Long et al. [2007] found out that these difficulties in dealing with noise were not due solely to its exponential weighting scheme, but were inherent in the potential function approach to boosting.

4.1.4.2.3 Random classification noise Random classification noise (RCN) is a well studied model of how benign noise can affect data. Given a multiset S of labelled examples and a value $0 < \nu < \frac{1}{2}$, let let $\mathcal{D}_{\nu,S}$ denote the distribution corresponding to S which is corrupted with random classification noise at rate ν. A draw from $\mathcal{D}_{\nu,S}$ is obtained by drawing (x, y) uniformly at random from S and independently flipping the binary label y with probability ν.
We say that an algorithm B is a boosting algorithm which tolerate RCN at rate ν if B has the following property: Given c the target classifier, S a multiset of m examples, and $\mathcal{H}$ the above set of base classifiers. Then for any $\epsilon > 0$,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

if B is run with $\mathcal{H}$ as the set of base classifiers on $\mathcal{D}_{\nu,S}$, at some stage of boosting B constructs a classifier g with accuracy

$$\frac{|\{(x_i, y_i) \in S : \ g(x_i) = y_i\}|}{m} \geqslant 1 - \nu - \epsilon$$

It has been shown that no boosting algorithm could be guaranteed to construct a classifier g whose accuracy exceeds $1 - \nu$ in the presence of RCN at rate ν (see Kalai et al. [2005]). However, there are known boosting algorithms that tolerate RCN at rate ν for $0 < \nu < \frac{1}{2}$. They do not follow the convex potential function approach, but build a branching program over the base classifiers.
Long et al. [2007] showed that no convex potential function booster could have this property. They showed that for any such booster and any non-zero random classification noise rate ν, there is a simple data set of examples which is efficiently learnable by such a booster if there is no noise, but which cannot be learned to accuracy better than $\frac{1}{2}$ if there is random classification noise at rate ν. More formally, we have the following:

Theorem 4.1.1 *Fix any convex potential function ϕ and any noise rate $0 < \nu < \frac{1}{2}$. Then*

1. *The global-minimising booster B_{ϕ}^{ideal} does not tolerate RCN at rate ν.*
2. *For any number K of rounds, the early stopping regularised booster $B_{\phi,K}^{early}$ does not tolerate RCN at rate ν.*
3. *For any $C > 0$, the L_1-regularised booster $B_{\phi,C}^{L_1}$ does not tolerate RCN at rate ν.*

4.1.5 Random forest

Random forests (RF), or random decision forests, are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees (see Ho [1995] [1998]). The algorithm was extended by Breiman [2001]. Another extension combines the bagging idea from Breiman and random selection of features, introduced first by Ho and later independently by Amit et al. [1997], in order to construct a collection of decision trees with controlled variance.
The idea of ensemble learning is that a single classifier is not sufficient for determining class of test data since the latter is not able to distinguish between noise and pattern. Thus, random decision forests correct for decision trees' habit of overfitting to their training set. Ho established that forests of trees splitting with oblique hyperplanes can gain accuracy as they grow without suffering from overtraining, as long as the forests are randomly restricted to be sensitive to only selected feature dimensions. He eventually concluded that other splitting methods behave similarly, as long as they are randomly forced to be insensitive to some feature dimensions. Note that this observation of a more complex classifier (a larger forest) getting more accurate nearly monotonically is in sharp contrast to the common belief that the complexity of a classifier can only grow to a certain level of accuracy before being hurt by overfitting.

4.1.5.1 Tree bagging

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, ..., x_m$ with responses $Y = y_1, ..., y_m$, bagging repeatedly (N times) selects a random sample with replacement of the training set and fits trees to these samples.

Algorithm 5 Tree Bagging

```
for n = 1 to N do
    sample, with replacement, n training examples from X, Y: called X_n, Y_n
    train a classification or regression tree f_n on X_n, Y_n
end for
```


This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

After training, predictions for unseen samples $x^{'}$ can be made by averaging the predictions from all the individual regression trees on $x^{'}$

$$\widehat{f} = \frac{1}{N} \sum_{n=1}^{N} f_n(x^{'})$$

or by taking the majority vote in the case of classification trees.
This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Bootstrap sampling is a way of de-correlating the trees by showing them different training sets. An estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on $x^{'}$

$$\sigma^2 = \frac{1}{N-1} \sum_{n=1}^{N} \left(f_n(x^{'}) - \widehat{f}\right)^2$$

An optimal number of trees N can be found using cross-validation, or by observing the out-of-bag error.

4.1.5.2 The procedure

Random forests (RF) is an extension of bagging incorporating randomised feature selection. During the construction of a component decision tree, at each step of split selection (deterministic), RF first randomly selects a subset of features, and then carries out the conventional split selection procedure within the selected feature subset. That is, RF use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. It is called feature bagging. Typically, for a classification problem with p features, $\sqrt{p}$ features are used in each split.
The parameter K controls the incorporation of randomness. When K equals the total number of features, the constructed decision tree is identical to the traditional deterministic decision tree, but when $K = 1$, a feature will be selected randomly. Breiman [2001] suggested $K = \log p$, the logarithm of the number of features.

Algorithm 6 Random tree algorithm

Require: Input: data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$, feature subset size K
$N \leftarrow$ create a tree node based on D
if all instances in the same class **then**
  return N
end if
$\mathcal{F} \leftarrow$ the set of features that can be split further
if $\mathcal{F}$ is empty **then**
  return N
end if
$\widetilde{\mathcal{F}} \leftarrow$ select K features from $\mathcal{F}$ randomly
$N \cdot f \leftarrow$ the feature that has the best split point in $\widetilde{\mathcal{F}}$
$N \cdot p \leftarrow$ the best split point on $N \cdot f$
$D_l \leftarrow$ subset of D with values of $N \cdot f$ smaller than $N \cdot p$
$D_r \leftarrow$ subset of D with values of $N \cdot f$ no smaller than $N \cdot p$
$N_l \leftarrow$ call the process with parameters (D_l, K)
$N_r \leftarrow$ call the process with parameters (D_r, K)
return N
Require: Output: a random decision tree

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Adding one further step of randomisation yields extremely randomized trees, or ExtraTrees. These are trained using bagging and the random subspace method, like in an ordinary random forest, but additionally the top-down splitting in the tree learner is randomised. Instead of computing the locally optimal feature/split combination (based on, information gain or the Gini impurity), for each feature under consideration, a random value is selected for the split. This value is selected from the feature's empirical range (in the tree's training set, that is, the bootstrap sample).
The technique is as follows: The first step in measuring the variable importance in a data set $D_n = \{(X_i, Y_i)\}_{i=1}^n$ is to fit a random forest to the data. During the fitting process the out-of-bag error for each data point is recorded and averaged over the forest (errors on an independent test set can be substituted if bagging is not used during training). To measure the importance of the jth feature after training, the values of the jth feature are permuted among the training data and the out-of-bag error is again computed on this perturbed data set. The importance score for the jth feature is computed by averaging the difference in out-of-bag error before and after the permutation over all trees. The score is normalised by the standard deviation of these differences.
As an example, we consider a random forest that performs sampling with replacement, such that, given N trees that are to be learnt, the results are based on these data set samples. Each tree is learnt using 3 features selected randomly. After the creation of n trees, when testing data is used, the decision which majority of trees come up with is considered as the final output. This approach also avoids problem of overfitting. The implementation is summarised in the Algorithm (7).

Algorithm 7 Random Forest

Require: Input: training set D, number of trees in the ensemble N
Require: Output: a composite model M_*

```
for n = 1 to N do
    create bootstrap sample D_n by sampling D with replacement
    select three features randomly
    use D_n and randomly select three features to derive tree M_n
end for
return M_*
```

4.1.6 Gradient boosting

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models (see Friedman [2001]). When a decision tree is the weak learner, the resulting algorithm is called gradient boosted trees, which usually outperforms random forest (see Hastie et al. [2009]). It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimisation of an arbitrary differentiable loss function.

4.1.6.1 Description

Gradient boosting combines weak learners into a single strong learner in an iterative fashion. We use the training set $D = \{(x_1, y_1), ..., (x_n, y_n)\}$ of known values of $x_i \in \mathcal{X}$ and corresponding values $y_i \in \mathcal{Y}$. For ease of exposition, we consider the least-squares regression setting, where the goal is to teach a model F to predict values of the form $\widehat{y} = F(x)$ by minimising the mean square error $\frac{1}{n}\sum_i(\widehat{y}_i - y_i)^2$ (see Section (11.1.2)) for some training set of size n. Rather that letting the model F be a function of the parameters W as with neural networks (see Section (5.1)), gradient boosting starts with a very simple model F_0 and search for the best model F_m minimising the MSE in an iterate process. We consider a gradient boosting algorithm with M stages. At each stage m, $1 \leqslant m \leqslant M$ of gradient boosting, we assume some imperfect model F_m, such as the mean of y. In order to improve F_m, the algorithm add some new estimator $h_m(x)$. Thus, the iterative process satisfies

$$F_{m+1}(x) = F_m(x) + h_m(x) = y$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

or equivalently

$$h_m(x) = y - F_m(x)$$

As a result, the gradient boosting fit h_m to the residual $y - F_m(x)$. Here, each new model F_{m+1} attempts to correct the error of its predecessor F_m. A generalization of this idea to loss functions other than squared error, and to classification and ranking problems, follows from the observation that residuals $h_m(x)$ for a given model are the negative gradients of the mean squared error (MSE) loss function (with respect to $F(x)$)

$$\begin{aligned} L_{MSE} &= \frac{1}{2}(y - F(x))^2 \\ h_m(x) &= -\frac{\partial L_{MSE}}{\partial F}(x) = y - F(x) \end{aligned}$$

Hence, gradient boosting can be specialised to a gradient descent algorithm. As such, generalising it entails plugging in a different loss and its gradient.

***Remark* 4.1.1** *At each iteration, h_m is the difference of the delta rule in Equation (5.1.9) representing the scalar error between the prediction F_m and the target value y.*

As discussed in Section (10.1.1), in supervised learning, we want to find an approximation $F(x)$ to the actual value y by minimising the expected value of some specified loss function $L(y, F(x))$:

$$\widehat{F} = \arg\min_F E_{x,y}[L(y, F(x))]$$

In gradient boosting, we assume a real-valued y and seeks an approximation $\widehat{F}(x)$ in the form of a weighted sum of functions $h_i(x)$, $i = 1, ..., M$, from some class $\mathcal{H}$, called base (or weak) learners. That is,

$$\widehat{F}(x) = \sum_{i=1}^{M} \gamma_i h_i(x) + c$$

where γ_i is the ith weight, c is a constant. In accordance with the empirical risk minimisation (ERM) principle, the method tries to find an approximation $\widehat{F}(x)$ minimising the average value of the loss function on the training set. It minimises the empirical risk by starting with a model consisting of a constant function $F_0(x)$, and incrementally expanding it in a greedy fashion:

$$\begin{aligned} F_0(x) &= \arg\min_\gamma \sum_{i=1}^{n} L(y_i, \gamma) \\ F_m(x) &= F_{m-1}(x) + \arg\min_{h_m \in \mathcal{H}} \Big[\sum_{i=1}^{n} L\big(y_i, F_{m-1}(x_i) + h_m(x_i)\big)\Big] \end{aligned}$$

for some class of base learner $\mathcal{H}$.

4.1.6.2 Implementation

However, in general, choosing the best function h at each step for an arbitrary loss function L is a computationally infeasible optimisation problem. We must therefore consider a simplified version of the problem by applying a steepest descent step (see Section (18.3.1.1)) to this minimisation problem. Assuming the continuous case, where $\mathcal{H}$ is the set of arbitrary differentiable functions on $\mathbb{R}$, we can update the model as follows:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$
\begin{aligned}
F_m(x) &= F_{m-1}(x) - \gamma_m \sum_{i=1}^{n} \nabla_{F_{m-1}} L\big(y_i, F_{m-1}(x_i)\big) \\
\gamma_m &= \arg\min_{\gamma} \sum_{i=1}^{n} L\Big(y_i, F_{m-1}(x_i) - \gamma \nabla_{F_{m-1}} L\big(y_i, F_{m-1}(x_i)\big)\Big)
\end{aligned}
$$

where the derivatives are taken with respect to the functions F_i for $i \in \{1, ..., m\}$, and γ_m is the step length.
In the discrete case, when the set $\mathcal{H}$ is finite, we choose the candidate function h closest to the gradient of L for which the coefficient γ may be calculated with the help of line search algorithm. This is a heuristic approach which does not yield an exact solution to the given problem, but rather an approximation. The implementation is summarised in the Algorithm (8) (see Hastie et al. [2009]).

Algorithm 8 Random Boosting

Require: Input: training set $D = \{(x_i, y_i); i = 1, ..., n\}$, a differentiable loss function L, number of iterations M
Require: Output: the model $F_m(x)$

1: Initialise model with a constant value

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

2: **for** $m = 1$ **to** M **do**
3: Compute the pseudo residuals

$$r_{i,m} = -\Big[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\Big]_{F(x)=F_{m-1}(x)} \text{ for } i = 1, ..., n$$

4: Fit a base learner $h_m(x)$ to pseudo residuals. Train it using the set $D_r = \{(x_i, r_{i,m}); i = 1, ..., n\}$.
5: Compute the multiplier γ_m by solving the one-dimensional optimisation problem

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\big(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\big)$$

6: Update the model

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

7: **end for**
8: **return** $F_m(x)$

An important part of gradient boosting method is regularisation by shrinkage which consists in modifying the update rule as follows:

$$F_m(x) = F_{m-1}(x) + \nu \gamma_m h_m(x) \,, 0 < \nu \leqslant 1$$

where ν is the learning rate. Using small learning rates (such as $\nu < 0.1$) yields dramatic improvements in the models' generalisation ability over gradient boosting without shrinking ($\nu = 1$), but at the cost of increasing computational time both during training and querying as lower learning rate requires more iterations.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.1.6.3 Gradient tree boosting

In general, gradient boosting is used with decision trees (especially CART trees) of a fixed size as base learners. In that setting, Friedman proposed a modification to gradient boosting method which improves the quality of fit of each base learner. We consider a gradient boosting fitting a decision tree $h_m(x)$ at the mth step and let J_m be the number of its leaves. The tree partitions the input space into J_m disjoint regions $R_{1,m}, ..., R_{J_m,m}$ and predicts a constant value in each region. The output of the tree $h_m(x)$ for input x can be written as

$$h_m(x) = \sum_{j=1}^{J_m} b_{j,m} I_{\{R_{j,m}\}}(x)$$

where I is the indicator function and $b_{j,m}$ is the value predicted in the region $R_{j,m}$. Then, the coefficients $b_{j,m}$ are multiplied by some value γ_m chosen using line search so as to minimise the loss function. The model is updated as follows:

$$\begin{aligned} F_m(x) &= F_{m-1}(x) + \gamma_m h_m(x) \\ \gamma_m &= \arg\min_{\gamma} \sum_{i=1}^{n} L\big(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\big) \end{aligned}$$

The author modified this algorithm so that it chooses a separate optimal value $\gamma_{j,m}$ for each region of the tree, instead of a single γ_m for the whole tree. This is called the TreeBoost. The coefficients $b_{j,m}$ from the tree-fitting procedure can then be discarded and the update rule becomes

$$\begin{aligned} F_m(x) &= F_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{j,m} I_{\{R_{j,m}\}}(x) \\ \gamma_{j,m} &= \arg\min_{\gamma} \sum_{x_i \in R_{j,m}} L\big(y_i, F_{m-1}(x_i) + \gamma\big) \end{aligned}$$

Fitting the training set too closely can lead to degradation of the model's generalisation ability. One natural regularization parameter is the number of gradient boosting iterations M. Increasing M reduces the error on training set, but setting it too high may lead to overfitting. An optimal value of M is often selected by monitoring prediction error on a separate validation data set. Another regularisation parameter is the depth of the trees. The higher this value the more likely the model will overfit the training data. Regularisation can also be achieved by limiting the minimum number of observations in trees' terminal nodes. It is used in the tree building process by ignoring any splits that lead to nodes containing fewer than this number of training set instances. Imposing this limit helps to reduce variance in predictions at leaves.

4.1.7 Towards combination methods

Ensemble methods resort to combination to achieve a strong generalisation ability, where the combination method plays a crucial role. Some of the reasons for improvement are

- Statistical issue: high variance
- Computational issue: high computational variance
- Representational issue: high bias

Through combination, the variance as well as the bias of learning algorithms may be reduced.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.1.7.1 Averaging

Averaging is the most fundamental combination method for numeric outputs. To illustrate the method, we take regression as an example. Thus, we consider the regression setting and suppose that we estimate N models (learners) $\{\mathcal{M}_1, ..., \mathcal{M}_N\}$, with $h_i = \mathcal{M}_i$, $i = 1, ..., N$, on a single data set D leading to corresponding predictions $h_1(x), ..., h_N(x)$ with instance x (or feature vector) where $h_i(x) = \widehat{f}_i(x) \in \mathbb{R}$. In model averaging, we compute the prediction H by directly averaging the outputs of individual learners

$$H(x) = \widehat{f}_{ave}(x) = \sum_{i=1}^{N} \omega_i h_i(x) \tag{4.1.1}$$

where ω_i, $i = 1, .., N$, are fixed model weights. There are several ways for choosing the weights ω_i, $i = 1, .., N$.

4.1.7.1.1 Simple averaging The simplest one being to set $\omega_i = \frac{1}{N}$ to get a simple model average. Suppose the underlying true function we try to learn is $f(x)$, and x is sampled according to a distribution $p(x)$. The output of each learner can be written as the true value plus an error item as follows:

$$h_i(x) = f(x) + \epsilon_i(x)\,, i = 1, ..., N$$

Then, the mean squared error of h_i can be written as

$$\mathcal{E}(h) = \int \big(h_i(x) - f(x)\big)^2 p(x)dx = \int \epsilon_i^2(x)p(x)dx$$

and the averaged error made by the individual learners is

$$\overline{\mathcal{E}}(h) = \frac{1}{N}\sum_{i=1}^{N} \int \epsilon_i^2(x)p(x)dx$$

Similarly, the expected error of the combined learner is

$$\mathcal{E}(H) = \int \Big(\frac{1}{N}\sum_{i=1}^{N} h_i(x) - f(x)\Big)^2 p(x)dx = \int \big(\frac{1}{N}\sum_{i=1}^{N} \epsilon_i(x)\big)^2 p(x)dx$$

Thus, one can see that

$$\mathcal{E}(H) \leqslant \overline{\mathcal{E}}(h)$$

which states that the expected ensemble error will be no larger than the averaged error of the individual learners. In addition, if we assume that the errors ϵ_i, $i = 1, ..., N$, have zero mean and are uncorrelated

$$\int \epsilon_i p(x)dx = 0 \text{ and } \int \epsilon_i \epsilon_j p(x)dx = 0 \text{ for } i \neq j$$

we get

$$\mathcal{E}(H) = \frac{1}{N}\overline{\mathcal{E}}(h)$$

so that the ensemble error is smaller by a factor of N than the averaged error of the individual learners.

***Remark* 4.1.2** *In general, the errors are typically highly correlated since the individual learners are trained on the same problem.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.1.7.1.2 Weighted averaging More generally, assume $\omega_i \geqslant 0$ and $\sum_{i=1}^N \omega_i = 1$, then

$$\begin{aligned}\mathcal{E}(H) &= \int\Big(\sum_{i=1}^N \omega_i h_i(x) - f(x)\Big)^2 p(x)dx \\ &= \int\Big(\sum_{i=1}^N \omega_i h_i(x) - f(x)\Big)\Big(\sum_{j=1}^N \omega_j h_j(x) - f(x)\Big)p(x)dx \\ &= \sum_{i=1}^N\sum_{j=1}^N \omega_i\omega_j C_{ij}\end{aligned}$$

where

$$C_{ij} = \int\big(h_i(x) - f(x)\big)\big(h_j(x) - f(x)\big)p(x)dx$$

and C is the correlation matrix. Thus, the optimal weights can be solved by

$$\omega = \arg\min_{\omega} \sum_{i=1}^N\sum_{j=1}^N \omega_i\omega_j C_{ij}$$

Applying the Lagrange multiplier method, and assuming the correlation matrix C to be invertible, we get

$$\omega_i = \frac{\sum_{j=1}^N C_{ij}^{-1}}{\sum_{k=1}^N\sum_{j=1}^N C_{kj}^{-1}}$$

which provides a closed-form solution to the optimal weights.

***Remark* 4.1.3** *The correlation matrix is usually singular or ill-conditioned, since the errors of the individual learners are typically highly correlated and many individual learners may be similar since they are trained on the same problem.*

Therefore ω_i is generally infeasible, and moreover, it does not guarantee non-negative solutions.
Note, voting, is special cases or variants of weighted averaging. More generally any ensemble method can be regarded as trying a specific way to decide the weights for combining the individual learners, and different ensemble methods can be regarded as different implementations of weighted averaging.
For example, given a training set with m number of training examples, we can choose the coefficients by least squares

$$\min_{\omega_i : i\in[1,N]} \sum_{k=1}^m \big(y_k - \hat{f}_{ave}(x_k)\big)^2$$

where we may want to impose restrictions on the weights such as non-negativity and a total sum of one. However, the least squares estimation might not work well since the regression models $\{h_1(\cdot), ..., h_N(\cdot)\}$ were previously estimated based on the same training data.
The minimisation does not take into account the complexity of the individual models, and will tend to overfit by putting too much weight on the most complex models (which will have low training errors).

***Remark* 4.1.4** *When the data is noisy and insufficient, the estimated weights are often unreliable.*

In general, simple averaging is appropriate for combining learners with similar performances, whereas if the individual learners exhibit non-identical strength, weighted averaging with unequal weights may achieve a better performance.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.1.7.2 Voting

Voting is the most popular and fundamental combination method for nominal outputs. Thus, to illustrate the method, we consider classification and suppose given a set of N individual classifiers $\{h_1, ..., h_N\}$. We want to combine the h_i to predict the class label from a set of l possible class labels $\{c_1, ..., c_l\}$. We assume that for any instance x, the outputs of the classifier h_i are given as an l-dimensional label vector $(h_{i,1}(x), ..., h_{i,l}(x))^\top$, where $h_{i,j}(x)$ is the output of h_i for the class label c_j. The output can take different types of values according to the information provided by the individual classifiers. For instance:

- Crisp label: $h_{i,j}(x) \in \{0, 1\}$ which takes value one if h_i predicts c_j as the class label and zero otherwise.
- Class probability: $h_{i,j}(x) \in [0, 1]$ which is an estimate of the posterior probability $P(c_j|x)$.

For classifiers that produce un-normalised margins, such as SVMs, calibration methods such as Platt scaling or Isotonic Regression can be used to convert such an output to a probability.

4.1.7.2.1 Majority voting Majority voting (MV) is the most popular voting method. Every classifier votes for one class label, and the final output class label is the one that receives more than half of the votes; if none of the class labels receives more than half of the votes, a rejection option will be given and the combined classifier makes no prediction. The output class label of the ensemble is

$$H(x) = \begin{cases} c_j \text{ if } \sum_{i=1}^N h_{i,j}(x) > \frac{1}{2}\sum_{k=1}^l \sum_{i=1}^N h_{i,k}(x) \\ \text{rejection if a tax credit} \end{cases}$$

For example, given a total of N classifiers for a binary classification problem, the ensemble decision will be correct if at least $\lfloor \frac{N}{2} + 1 \rfloor$ classifiers choose the correct class label.
Assume that the outputs of the classifiers are independent and each classifier has an accuracy p [3]. Then, the probability of the ensemble for making a correct decision can be calculated using a binomial distribution (see Hansen et al. [1990]). The probability of obtaining at least $\lfloor \frac{N}{2} + 1 \rfloor$ correct classifiers out of N is

$$P_{mv} = \sum_{k=\lfloor \frac{N}{2}+1 \rfloor}^{N} \binom{N}{k} p^k (1-p)^{N-k}$$

Lam et al. [1997] showed that

- if $p > \frac{1}{2}$, then P_{mv} is monotonically increasing in N, and
$$\lim_{N\to\infty} P_{mv} = 1$$
- if $p < \frac{1}{2}$, then P_{mv} is monotonically decreasing in N, and
$$\lim_{N\to\infty} P_{mv} = 0$$
- if $p = \frac{1}{2}$ then $P_{mv} = \frac{1}{2}$ for any N

***Remark* 4.1.5** *This result is obtained based on the assumption that the individual classifiers are statistically independent, but in practice the classifiers are generally highly correlated since they are trained on the same problem.*

[3] each classifier makes a correct classification at probability p

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.1.7.2.2 Plurality voting Plurality voting (PV) takes the class label which receives the largest number of votes as the final winner. That is, the output class label of the ensemble is

$$H(x) = c_{\arg\max_j \sum_{i=1}^N h_{i,j}(x)}$$

and ties are broken arbitrarily. Note, plurality voting does not have a reject option. Further, in the case of binary classification, plurality voting coincides with majority voting.

4.1.7.2.3 Weighted voting If the individual classifiers have unequal performance, we want to give more power to the stronger classifiers in voting; this is realised by weighted voting. The output class label of the ensemble is

$$H(x) = c_{\arg\max_j \sum_{i=1}^N \omega_i h_{i,j}(x)}$$

where ω_i is the weight assigned to the classifier h_i. Again, we want $\omega_i > 0$ and $\sum_{i=1}^N \omega_i = 1$. Further, we need to decide how to obtain the weights. We let $\ell = (\ell_1, ..., \ell_N)^\top$ be the outputs of the individual classifiers, where ℓ_i is the class label predicted for the instance x by the classifier h_i and p_i denotes the accuracy of h_i. There is a Bayesian optimal discriminant function for the combined output on class label c_j

$$H_j(x) = \log\left(P(c_j)P(\ell|c_j)\right)$$

Assuming that the outputs of the individual classifiers are conditionally independent

$$P(\ell|c_j) = \prod_{i=1}^N P(\ell_i|c_j)$$

then one can show

$$H_j(x) = \log\left(P(c_j) + \sum_{i=1,\ell_i=c_j}^N \log\frac{p_i}{1-p_i} + \sum_{i=1}^N \log\left(1-p_i\right)\right.$$

Since $\sum_{i=1}^N \log\left(1-p_i\right)$ does not depend on the class label c_j, and $\ell_i = c_j$ can be expressed by the result of $h_{i,j}(x)$, the discriminant function can be simplified to

$$H_j(x) = \log\left(P(c_j) + \sum_{i=1}^N h_{i,j}(x)\log\frac{p_i}{1-p_i}\right.$$

The first term on the RHS does not rely on the individual learners, while the second term discloses that the optimal weights for weighted voting satisfy

$$\omega_i \propto \log\frac{p_i}{1-p_i}$$

which shows that the weights should be in proportion to the performance of the individual learners.

***Remark* 4.1.6** *The above formula obtained by assuming independence among the outputs of the individual classifiers, yet this does not hold since all the individual classifiers are trained on the same problem and they are usually highly correlated.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.1.7.2.4 Soft voting Soft voting (SV) is applied to classifiers which produce class probability outputs. The individual classifier h_i outputs a l-dimensional vector $(h_{i,1}(x), ..., h_{i,l}(x))^\top$ for the instance x, where $h_{i,j}(x) \in [0,1]$ can be regarded as an estimate of the posterior probability $P(c_j|x)$. In the case of uniform weighting, the simple soft voting average all the individual outputs, getting the final output for class c_j as

$$H_j(x) = \frac{1}{N}\sum_{i=1}^{N} h_{i,j}(x)$$

On the other hand, the weighted soft voting method can be any of the following three forms:

1. A classifier-specific weight is assigned to each classifier, and the combined output for class c_j is

$$H_j(x) = \sum_{i=1}^{N} \omega_i h_{i,j}(x)$$

   where ω_i is the weight of the classifier h_i.

2. A class-specific weight is assigned to each classifier per class, and the combined output for class c_j is

$$H_j(x) = \sum_{i=1}^{N} \omega_{i,j} h_{i,j}(x)$$

   where $\omega_{i,j}$ is the weight of the classifier h_i for the class c_j.

3. A weight is assigned to each example of each class for each classifier, and the combined output for class c_j is

$$H_j(x) = \sum_{i=1}^{N}\sum_{k=1}^{m} \omega_{i,j,k} h_{i,j}(x)$$

   where $\omega_{i,j,k}$ is the weight of the instance x_k of the class c_j for the classifier h_i.

Focussing on case (2), we have

$$h_{i,j}(x) = P(c_j|x) + \epsilon_{i,j}(x)$$

where $\epsilon_{i,j}(x)$ is the approximation error. In classification, the target output is given as a class label. In the case of an unbiased estimation the combined output $H_j(x)$ is also unbiased, and we can obtain a variance-minimised unbiased estimation of $H_j(x)$ for $P(c_j|x)$ by estimating the weights. Minimizing the variance of the combined approximation error $\sum_{i=1}^{N} \omega_{i,j}\epsilon_{i,j}(x)$ under the constraints $\omega_{i,j} \geqslant 0$ and $\sum_{i=1}^{N} \omega_{i,j} = 1$, we get the optimisation problem

$$\omega_j = \arg\min_{\omega_j} \sum_{k=1}^{m} \Big(\sum_{i=1}^{N} \omega_{i,j} h_{i,j}(x) - I(f(x_k) = c_j)\Big)^2, j = 1, ..., l$$

and solve for the weights. In general, soft voting is used for homogeneous ensembles. In the case of heterogeneous ensembles, one must perform a calibration where the class probability outputs are converted to class label outputs by setting $h_{i,j}(x) = 1$ if $h_{i,j}(x) = \max\{h_{i,j}(x)\}$ and 0 otherwise. The voting methods for crisp labels can then be applied.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.2 Online learning and regret-minimising algorithms

4.2.1 Simple online algorithms

First, we introduce some notation which will be used through out this section. We use relint (S) to refer to the relative interior of a convex set S, which is the set S minus all of the points on the relative boundary. We use closure (S) to refer to the closure of S, the smallest closed set containing all of the limit points of S. For any subset S of $\mathbb{R}^d$, we let $\mathcal{H}(S)$ denote the convex hull of S. We let $\Delta_n = \{x \in \mathbb{R}^n : \sum_i^n x_i = 1, x_i \geqslant 0 \forall i\}$ be the n-dimensional probability simplex.

4.2.1.1 The Halving algorithm

To introduce online learning we are first going to present the Halving algorithm where a player has access to the prediction of N experts denoted by

$$f_{1,t}, ..., f_{N,t} \in \{0, 1\}$$

At each time $t = 1, .., T$, we observe $f_{i,t}$, i=1,..,N, and predict $p_t \in \{0, 1\}$. We then observe $y_t \in \{0, 1\}$ and suffer loss $I_{\{p_t \neq y_t\}}$. Suppose $\exists j$ such that $f_{j,t} = y_t$ for all $t \in [T]$. Then, the Halving theorem predicts $p_t = majority(C_t)$, where $C_1 = [N]$ and $C_t \subseteq [N]$ is defined below for $t > 1$.

Theorem 4.2.1 *If $p_t = majority(C_t)$ and*

$$C_{t+1} = \{i \in C_t : f_{i,t} = y_t\}$$

then we will make at most $\log_2 N$ mistakes.

In fact, for every t at which there is a mistake, at least half of the experts in C_t are wrong, so that $|C_{t+1}| \leqslant \frac{|C_t|}{2}$. It follows that $|C_T| \leqslant \frac{|C_1|}{2^M}$ where M is the tolal number of mistakes. Further, since there is a perfect expert, $|C_T| \geqslant 1$. As a result, recalling that $C_1 = [N]$, then $1 \leqslant \frac{N}{2^M}$, such that, after rearranging we get $M \leqslant \log_2 N$.

4.2.1.2 The weighted majority algorithm

To illustrate online, no-regret learning, we consider the problem of learning from expert advice where an algorithm must make a sequence of predictions based on the advice of a set of N experts and receive a corresponding sequence of losses. The aim of the algorithm being to achieve a cumulative loss that is almost as low as the cumulative loss of the best performing expert in hindsight. No statistical assumptions are made about these losses. At every time step $t \in \{1, .., T\}$, every expert $i \in \{1, .., N\}$ predict $f_{i,t} \in [0, 1]$ and receives a loss $l_{i,t} = l(f_{i,t}, y_t) \in [0, 1]$ such that the cumulative loss of expert i at time T is given by $L_{i,T} = \sum_{t=1}^T l_{i,t}$. We let the algorithm $\mathcal{A}$ (or player) maintains a weight $w_{i,t}$ for each expert i at time t, where $\sum_{i=1}^N w_{i,t} = 1$. Hence, these weights can be seen as a distribution over the experts. The algorithm then receives its own instantaneous loss

$$l_{\mathcal{A},t} = \sum_{i=1}^N w_{i,t} l_{i,t}$$

which can be interpreted as the expected loss the algorithm would receive if it always chose an expert to follow according to the current distribution. The cumulative loss of $\mathcal{A}$ up to time T is defined as

$$L_{\mathcal{A},T} = \sum_{t=1}^T l_{\mathcal{A},t} = \sum_{t=1}^T \sum_{i=1}^N w_{i,t} l_{i,t}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

For simplicity of exposition we use l_t, L_t, and w_t to refer to the vector of losses, vector of cumulative loss, and vector of weights, respectively, for each expert on round t. We will use the dot product to relate these vectors. Since the algorithm will not achieve a small cumulative loss if none of the experts perform well, we generally measure the performance of an algorithm in terms of its regret, defined as the difference between the cumulative loss of the algorithm and the loss of the best performing expert

$$R_T = L_{\mathcal{A},T} - \min_{i \in \{1,\dots,N\}} L_{i,T}$$

Hence, the algorithm is said to have no-regret if the average per time step regret approaches 0 as $T \to \infty$. The Randomized Weighted Majority (WM) algorithm (see Littlestone et al. [1994]) is an example of a no-regret algorithm, where the Weighted Majority uses weights

$$w_{i,t} = \frac{e^{-\eta L_{i,t-1}}}{\sum_{j=1}^{N} e^{-\eta L_{j,t-1}}}$$

where $\eta > 0$ is a learning rate parameter and the player's predict is $p_t = \sum_{i=1}^{N} w_{i,t} f_{i,t}$. The pseudo code for the weighted majority algorithm is

1. for $t = 1, .., t$ do
    - player observes $f_{1,t}, .., f_{N,t}$
    - player predicts p_t
    - adversary reveals outcome y_t
    - player suffers loss $l(p_t, y_t)$
    - experts suffer loss $l(f_{i,t}, y_t)$
2. end for

After T trials, the regret of WM can be bounded as

$$R_T = L_{WM(\eta),T} - \min_{i \in \{1,\dots,N\}} L_{i,T} \leqslant \eta T + \frac{\log N}{\eta}$$

such that if T is known in advance, setting $\eta = \sqrt{\frac{\log N}{T}}$ yields the standard $\mathcal{O}(\sqrt{T \log N})$ regret bound. Further, setting $\eta = \sqrt{8 \frac{\log N}{T}}$ we get the bound

$$R_T \leqslant \sqrt{\frac{T}{2} \log N}$$

4.2.2 The online convex optimisation

4.2.2.1 The online linear optimisation problem

Given the probability simplex Δ_N, the pseudo code for the online linear optimisation is

1. for $t = 1, .., t$ do
    - player predicts $w_t \in \Delta_N$ (w_t is essentially a probability distribution)
    - adversary reveals $l_t \in \mathbb{R}^N$
    - player suffers loss $w_t.l_t$ where $l_{i,t} = l(f_{i,t}, y_t)$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. end for

The learner suffers regret

$$R_T = \sum_{t=1}^{T} w_t \cdot l_t - \min_{w \in \Delta_N} \sum_{t=1}^{T} w \cdot l_t$$

where $l_t(w) = w \cdot l_t$. Note, for the simplex, the distribution w will place all the probability on the best expert. That is, the experts setting is just a special case of the online linear optimisation, where the set $\mathcal{K}$ is the N-simplex Δ_n.

It was proved that the weights chosen by WM are precisely those minimising a combination of empirical loss and an entropic regularisation term (see Kivinen et al. [1997]). That is, the weight vector w_t at time t is the solution to the following minimisation problem

$$\min_{w \in \Delta_N} w.L_{t-1} - \frac{1}{\eta} H(w)$$

where L_{t-1} is the vector of cumulative loss at time $t-1$, Δ_N is the probability simplex, and $H(\cdot)$ is the entropy function (see Section (2.2.2))

$$H(w) = -\sum_{i=1}^{N} w_i \log(w_i)$$

***Remark* 4.2.1** *The entropy function acts as a regularisation function for the weight vector w.*

4.2.2.2 Considering Bergmen divergence

Given the online linear optimisation problem described in Section (4.2.2.1), we consider the minimisation problem for the weight vector w_{t+1}

$$w_{t+1} \arg\min_{w \in \mathcal{K}} \eta \sum_{s=1}^{t} l_t(w) + R(w)$$

for some convex function $R(\cdot)$. We define $\Phi_0(w) = R(w)$ and $\Phi_t(w) = \Phi_{t-1}(w) + \eta l_t(w)$. We also let $D_f(x, y)$ be the Bergman divergence between x and y with respect to the function f (see details in Appendix (**??**)).

Lemma 4.2.1 *Suppose $\mathcal{K} = \mathbb{R}^N$, then for any $u \in \mathcal{K}$ we get*

$$\eta \sum_{t=1}^{T} [l_t(w) - l_t(u)] = D_{\Phi_0}(u, w_1) - D_{\Phi_T}(u, w_{T+1}) + \sum_{t=1}^{T} D_{\Phi_T}(w_t, w_{t+1})$$

and we also get

$$\sum_{t=1}^{T} l_t(w) \leqslant \inf_{u \in \mathcal{K}} \Big[\sum l_t(u) + \eta^{-1} D_R(u, w_1)\Big] + \eta \sum_{t=1}^{T} D_{\Phi_t}(w_t, w_{t+1})$$

Now, suppose that $\nabla R(w_1) = 0$ and $\mathbb{R}^N = \mathcal{K}$, then

$$w_{t+1} = \arg\min_{w \in \mathbb{R}^N} \big[\eta l_t(w) + D_{\Phi_{t-1}}(w, w_t)\big]$$

and the two minimisation problems are equivalent. That is,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}\eta l_t(w) &= \Phi_t(w) - \Phi_{t-1}(w) \\ \eta l_t(w) + D_{\Phi_{t-1}}(w, w_t) &= \Phi_t(w) - \Phi_{t-1}(w) + D_{\Phi_{t-1}}(w, w_t)\end{aligned}$$

Assuming that the two equations are equivalent for $\tau \leqslant t$, and w minimises Φ_{t-1}

$$\begin{aligned}&\nabla_w D_{\Phi_{t-1}}(w, w_t) = \nabla_w \Phi_{t-1}(w, w_t) - \nabla_w \Phi_{t-1}(w_t) \\ &\nabla \Phi_t(w_{t+1}) = \nabla \Phi_{t-1}(w_t) = ... = \nabla \mathbb{R}(w_1) = 0\end{aligned}$$

thus, $w_{t+1} = \arg\min_{w \in \mathcal{K}} \Phi_t(w)$. Assuming l_t's are linear functions, and letting R^* be the Legendre dual of the function $R(\cdot)$ (see details in Appendix (**??**)), we get

***Corollary* 3** *1.* $\eta\big(\sum l_t.w_t - \sum l_t \cdot u\big) = D_R(u, w_1) + D_R(u, w_{t+1}) + \sum D_R(w_t, w_t - 1)$ *for any* $u \in \mathbb{R}^N$

2. $w_{t+1} = \nabla R^*(\nabla R(w_t) - \eta l_t)$

Recall, the online gradient descent is $w_{t+1} = w_t - \eta l_t$. Further, if $R = \frac{1}{2}\|\cdot\|^{\frac{1}{2}}$, $\nabla R(w) = w$, $\nabla R^*(w) = w$, and if $l_t(\cdot)$ are convex (but not necessary linear), then

Lemma 4.2.2 *If we choose* $w_{t+1} = \arg\min_{w \in \mathbb{R}^N} \big[\eta \nabla l_t(w)^\top w + D_R(w, w_t)\big]$ *(or equivalently* $w_{t+1} = \arg\min_{w \in \mathbb{R}^N} \eta \sum \big[\nabla l_t(w)^\top w + R(x)\big]$*), then*

$$\sum_{t=1}^{T} \big[l_t(w_t) - l_t(u)\big] \leqslant \eta^{-1} D_R(u, w_1) + \sum_{t=1}^{T} D_R(w_t, w_{t+1})$$

4.2.2.3 More on the online convex optimisation problem

The WM is an example of a broader class of algorithms collectively known as Follow the Regularized Leader (FTRL) algorithm (see Hazan et al. [2008]). The FTRL template can be applied to a wide class of learning problems falling under the general framework of Online Convex Optimisation (see Zinkevich [2003]). Other problems falling into this framework include online linear pattern classification, online Gaussian density estimation, and online portfolio selection. Further, the online linear optimisation problem is an extension of the expert setting where the weights w_t are chosen from a fixed bounded convex action space $\mathcal{K} \subset \mathbb{R}^N$ (see Rakhlin [2009]). The algorithm follows

1. Input: convex compact decision set $\mathcal{K} \subset \mathbb{R}^N$.
2. Input: strictly convex differentiable regularisation function $\mathcal{R}(\cdot)$ defined on $\mathcal{K}$.
3. Parameter: $\eta > 0$.
4. Initialise: $L_1 =< 0, .., 0 >$.
5. for $t = 1, .., T$ do
6. The learner selects action $w_t \in \mathcal{K}$ according to

$$w_t = \arg\min_{w \in \mathcal{K}} w.L_{t-1} + \frac{1}{\eta}\mathcal{R}(w)$$

7. Nature reveals l_t, learner suffers loss $l_t.w_t$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

8. The learner updates $L_t = L_{t-1} + l_l$
9. end for

Abernethy et al. [2012] showed that the FTRL algorithms for online learning and the problem of pricing securities in a prediction market have a strong syntactic correspondence. To do so they needed to make two assumptions

***Assumption* 1** *For each time step t,* $\|l_t\| \leqslant 1$.

***Assumption* 2** *The regulasier* $\mathcal{R}(\cdot)$ *has the Legendre property (see Cesa-Bianchi et al. [2006]).* $\mathcal{R}$ *is strictly convex on relint* $(\mathcal{K})$ *and* $||\nabla\mathcal{R}(w)|| \to \infty$ *as* $w \to$ *relint* $(\mathcal{K})$.

As a result, the solution to the above algorithm will always occur in the relative interior of $\mathcal{K}$ such that the optimisation is unconstrained.

4.3 Combining by learning

4.3.1 Stacking

The main idea of stacking is that we want to learn whether training data have been properly learned. Thus, a learner is trained to combine the individual learners. The individual learners are called the first-level learners, while the combiner is called the second-level learner, or meta-learner. Given the first-level learners we generate a new data set for training the second-level learner, where the outputs of the first-level learners are regarded as input features while the original labels are still regarded as labels of the new training data.
There are two approaches for combining models: voting and stacking. The differences between the stacking and voting framework are as follow:

- In contrast to stacking, no learning takes place at the meta-level when combining classifiers by a voting scheme.
- Label that is most often assigned to a particular instance is chosen as the correct prediction when using voting.

Practically, stacking is concerned with combining multiple classifiers generated by different learning algorithms $L_1, ..., L_N$ on a single dataset D, which is composed by a feature vector $d_i = (x_i, y_i)$. The stacking process can be broken into two phases:

1. Generate a set of base-level classifiers $h_1, ..., h_N$ where $h_n = L_n(D)$.
2. Train a meta-level classifier to combine the outputs of the base-level classifiers.

The implementation is summarised in the Algorithm (9).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 9 Stacking procedure

Require: Input: data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$, first level learning algorithms $L_1, ..., L_N$, second level learning algorithm L
Require: Output: a composite model M_*
for $n = 1$ **to** N **do**
  $h_n = L_n(D)$ train a first-level learner by applying the first-level learning algorithm L_n
end for
$D^{'} = \varnothing$ generate a new data set
for $i = 1$ **to** m **do**
  for $n = 1$ **to** N **do**
    $z_{i,n} = h_n(x_i)$ train a first-level learner by applying the first-level learning algorithm L_n
  end for
  $D^{'} = D^{'} \cup \big((z_{i,1}, ..., z_{i,N}), y_i\big)$
end for
$h^{'} = L(D^{'})$ train the second-level learner $h^{'}$ by applying the second-level learning algorithm L to the new data set $D^{'}$
Require: Output: $H(x) = h^{'}\big(h_1(x), ..., h_N(x)\big)$

To avoid overfitting, it is suggested that the instances used for generating the new data set are excluded from the training examples for the first-level learners, and a cross validation or leave-one-out procedure is often recommended. The stacking framework is as follows:

- The training set for the meta-level classifier is generated through a leave-one-out cross validation process

$$\forall i = 1, ..., m \text{ and } \forall n = 1, ..., N \text{ then } h_n^i = L_n(D - d_i)$$

- The learned classifiers are then used to generate predictions for $d_i : z_i^n = h_n^i(x)$
- The meta-level dataset consists of examples of the form $\Big(\big(z_i^1, ..., z_i^N\big), y_i\Big)$, where the features are the predictions of the base-level classifiers and the class is the correct class of the example in hand.

In the case of k-fold cross-validation, the original training data set D is randomly split into k almost equal parts $D_1, ..., D_k$. Let D_j and $D_{(-j)} = D \backslash D_j$ to be the test and training sets for the jth fold. Given N learning algorithms, a first-level learner $h_n^{(-j)}$ is obtained by using the nth learning algorithm on $D_{(-j)}$. For x_i in D_j, we let $z_{i,n}$ denotes the output of the learner $h_n^{(-j)}$ on x_i. Then, at the end of the entire cross-validation process, the new data set is generated from the N individual learners as

$$D^{'} = \{\big(z_{i,1}, ..., z_{i,N}, y_i\big)\}_{i=1}^m$$

It is then applied to the second-level learning algorithm and the resulting learner $h^{'}$ is a function of $(z_1, ..., z_N)$ for y. Similarly to averaging described above, stacking is equivalent to solving the minimisation problem

$$\min_{\omega_m : m \in [1,M]} \sum_{i=1}^N \big(y_i - \widehat{f}_{avecv}(x_i)\big)^2 \tag{4.3.2}$$

where

$$\widehat{f}_{avecv}(x) = \sum_{m=1}^M \omega_m \widehat{f}_m^{(cv)}(x) \tag{4.3.3}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

and $\widehat{f}_m^{(cv)}(\cdot)$ are cross validation predictions. That is, model stacking fits a linear regression model based on constructed predictors derived from different models. We say that the linear regression model is therefore the meta model for the stack.

***Remark* 4.3.1** *More generally, we have*

$$\widehat{f}_{avecv}(x) = g\big(\widehat{f}_1^{(cv)}(x), ..., \widehat{f}_M^{(cv)}(x)\big)$$

There is no reason for $g(\cdot)$ to be linear, except for simplicity.

This idea can be generalised and used with any algorithm as a meta model. For instance, it can be applied to classification. Some options are:

- Hard voting classifier: (weighted) majority voting across several models.
- Soft voting classifier: (weighted) probability averages.
- Model stacking: fit a meta model such as a logistic regression using cross validation predictions from multiple models as inputs.

4.3.2 Infinite ensemble

Since most ensemble methods exploit only a small finite subset of hypotheses, Lin et al. [2008] developed an infinite ensemble framework that constructs ensembles with infinite hypotheses. It is based on support vector machines (SVM) and corresponds to learning the combination weights for all possible hypotheses. By embedding infinitely many hypotheses into a kernel, it can be found that the learning problem reduces to an SVM training problem with specific kernels.
Let $\mathcal{H} = \{h_\alpha : \alpha \in \mathcal{C}\}$ denote the hypothesis space, where $\mathcal{C}$ is a measure space. The kernel that embeds $\mathcal{H}$ is defined as

$$K_{\mathcal{H},r}(x_i, x_j) = \int_{\mathcal{C}} \Phi_{x_i}(\alpha)\Phi_{x_j}(\alpha)d\alpha$$

where $\Phi_x(\alpha) = r(\alpha)h_\alpha(x)$ and $r : \mathcal{C} \to \mathbb{R}_+$ is chosen such that the integral exists for all x_i, x_j. α denotes the parameter of the hypothesis h_α, and $Z(\alpha)$ means that Z depends on α. It has been proved that the kernel is valid. Following SVM, the framework formulates the following (primal) problem:

$$\begin{aligned}
&\min_{\omega \in L_2(\mathcal{C}), b \in \mathbb{R}, \xi \in \mathbb{R}^m} \frac{1}{2}\int_{\mathcal{C}} \omega^2(\alpha)d\alpha + C\sum_{i=1}^m \xi_i \\
&\text{s.t. } y_i\Big(\int_{\mathcal{C}} \omega(\alpha)r(\alpha)h_\alpha(x)d\alpha + b\Big) \geqslant 1 - \xi_i \\
&\xi_i \geqslant 0\,, \forall i = 1, ..., m
\end{aligned}$$

The final classifier obtained from this optimization problem is

$$g(x) = \text{sgn}\Big(\int_{\mathcal{C}} \omega(\alpha)r(\alpha)h_\alpha(x)d\alpha + b\Big)$$

By using the Lagrangian multiplier method and the kernel trick, the dual problem of the primal problem can be obtained, and the final classifier can be written in terms of the kernel $K_{\mathcal{H},r}$ as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$g(x) = \text{sgn}\big(\sum_{i=1}^{m} y_i \lambda_i K_{\mathcal{H},r}(x_i, x) + b\big)$$

where the λ_i are the Lagrange multipliers. By equivalence, the above equation infinite ensemble over $\mathcal{H}$. The learning problem can be reduced to solving an SVM with the kernel $K_{\mathcal{H},r}$ and the final ensemble can be obtained by applying typical SVM solvers.

4.3.3 Other methods

There are many other combination methods available and we are going to briefly present a few of them.

4.3.3.1 The algebraic methods

The algebraic methods which is based on a probabilistic framework (see Kittler et al. [1998]). Again, we let $h_{i,j}(x)$ be the class probability of c_j output from h_i. Bayesian decision theory states that given N classifiers, the instance x should be assigned to the class c_j which maximises the posteriori probability $P(c_j|h_{1,j}, ..., h_{N,j})$. From Bayes theorem, it follows that

$$P(c_j|h_{1,j}, ..., h_{N,j}) = \frac{P(c_j)P(h_{1,j}, ..., h_{N,j}|c_j)}{\sum_{i=1}^{l} P(c_i)P(h_{1,j}, ..., h_{N,j}|c_i)}$$

where $P(h_{1,j}, ..., h_{N,j}|c_j)$ is the joint probability distribution of the outputs from the classifiers. Assuming that the outputs are conditionally independent, that is, $P(h_{1,j}, ..., h_{N,j}|c_j) = \prod_{i=1}^{N} P(h_{i,j}|c_j)$, it follows that

$$\begin{aligned} P(c_j|h_{1,j}, ..., h_{N,j}) &= \frac{P(c_j)\prod_{i=1}^{N} P(h_{i,j}|c_j)}{\sum_{i=1}^{l} P(c_i)\prod_{k=1}^{N} P(h_{k,j}|c_i)} \\ &= P^{N-1}(c_j)\prod_{i=1}^{N} P(c_j|h_{i,j}) \end{aligned}$$

Since $h_{i,j}$ is the probability output, we have $P(c_j|h_{i,j}) = h_{i,j}$. Thus, if all classes are with equal prior, we get the product rule for combination

$$H_j(x) = \prod_{i=1}^{N} h_{i,j}(x)$$

Following the same approach, Kittler et al. [1998] derived soft voting method, as well as the maximum/minimum/median rules which choose the maximum/minimum/median of the individual outputs as the combined output. For instance, the median rule generates the combined output according to

$$H_j(x) = \underset{i}{\text{med}}\big(h_{i,j}(x)\big)$$

where $\text{med}(\cdot)$ is the median statistic.

4.3.3.2 Mixture of experts

Mixture of experts (ME) works in a divide-and-conquer strategy where a complex task is broken up into several simpler and smaller subtasks, and individual learners (called experts) are trained for different subtasks (see Jacobs et al. [1991], Xu et al. [1995]). Gating is usually employed to combine the experts. The individual learners are generated for different subtasks and there is no need to devote to diversity (see Figure (4.11)). Thus, the key problem

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

becomes to find the natural division of the task and then derive the overall solution from sub-solutions. To do so, we make the experts local by targeting each expert to a distribution specified by the gating function, rather than the whole original training data distribution.

Figure 4.11: Mixture of experts

As an example, we assume N experts and let the output y be a discrete variable with possible values 0 and 1 for binary classification. Given an input x, each local expert h_i tries to approximate the distribution of y and obtains a local output $h_i(y|x;\theta_i)$ where θ_i is the parameter of the ith expert h_i. The gating function provides a set of coefficients $\pi_i(x;\alpha)$ weighting the contributions of experts where α is the parameter of the gating function. Thus, the final output of the ME is a weighted sum of all the local outputs produced by the experts given by

$$H(y|x;\Psi) = \sum_{i=1}^{N} \pi_i(x;\alpha) \cdot h_i(y|x;\theta_i)$$

where Ψ includes all unknown parameters. The output of the gating function is often modelled by the softmax function as

$$\pi_i(x;\alpha) = \frac{e^{v_i^\top x}}{\sum_{i=1}^{N} e^{v_j^\top x}}$$

where v_i is the weight vector of the ith expert in the gating function, and α contains all the elements in the vector v_i. When training, $\pi_i(x;\alpha)$ states the probability of the instance x appearing in the training set of the ith expert h_i. However, when testing, it defines the contribution of h_i to the final prediction.
The training procedure tries to achieve two goals: for given experts, to find the optimal gating function; for given gating function, to train the experts on the distribution specified by the gating function. The unknown parameters are usually estimated by the Expectation Maximisation (EM) algorithm (see Jordan et al. [1995], Xu et al. [1996]). Hierarchical mixture of experts (HME) extends mixture of experts (ME) into a tree structure where the experts are built from multiple levels of experts and gating functions (see Jordan et al. [1992]).

4.3.4 Diversity measures

A symmetric measure will keep the same when the values of 0 (incorrect) and 1 (correct) in binary classification are swapped.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.3.4.1 Pairwise measures

When measuring diversity, the simplest approach is to measure the pairwise similarity/dissimilarity between two learners, and then average all the pairwise measurements to compute the overall diversity. One advantage of pairwise measures is that they can be visualized in $2d$-plots.
Given the data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$, we consider binary classification and define the contingency table (see Table (4.1)) for two classifiers h_i and h_j, where $a + b + c + d = m$ are non-negative variables showing the numbers of examples satisfying the conditions specified by the corresponding rows and columns. We are going to present a few representative pairwise measures based on these variables.

Classifiers	$h_i = +1$	$h_i = -1$
$h_j = +1$	a	c
$h_j = -1$	b	d

Table 4.1: Contingency table

The disagreement measure (see Skalak [1996]) between h_i and h_j is defined as the proportion of examples on which two classifiers make different predictions

$$\underset{ij}{\text{dis}} = \frac{b+c}{m}$$

where dis_{ij} takes values in $[0, 1]$. The larger the value, the larger the diversity.
The Q-statistic (see Yule [1900]) of h_i and h_j is defined as

$$Q_{ij} = \frac{ad - bc}{ad + bc}$$

where Q_{ij} takes value in the range $[-1, 1]$. It is zero if h_i and h_j are independent, it is positive if h_i and h_j make similar predictions, otherwise it is negative.
The correlation coefficient (see Sneath et al. [1973]) of h_i and h_i is defined as

$$\rho_{ij} = \frac{ad - bc}{\sqrt{(a+b)(a+c)(c+d)(b+d)}}$$

which is classical when measuring the correlation between two binary vectors. Note, ρ_{ij} and Q_{ij} have the same sign, and $|\rho_{ij}| \geqslant |Q_{ij}|$.
The kappa-statistic (see Cohen [1960]) is defined as

$$\kappa_p = \frac{\Theta_1 - \Theta_2}{1 - \Theta_2}$$

where Θ_1 and Θ_2 are the probabilities that the two classifiers agree and agree by chance, respectively. The probabilities for h_i and h_j, on the data set D, are estimated as follows:

$$\begin{aligned}\Theta_1 &= \frac{a+d}{m} \\ \Theta_2 &= \frac{(a+b)(a+c) + (c+d)(b+d)}{m^2}\end{aligned}$$

Then, $\kappa_p = 1$ if the two classifiers totally agree on D, $\kappa_p = 0$ if they agree by chance, and $\kappa_p < 0$ occurs when the agreement is even less than what is expected by chance. All these measures do not require knowing the classification correctness.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.3.4.2 Non-pairwise measures

We now focus on measuring the ensemble diversity directly rather than averaging pairwise measurements. We consider given a set of individual classifiers $\{h_1, ..., h_N\}$ and a data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$ where x_i is an instance and $y_i \in \{-1, +1\}$ is class label.
The Kohavi-Wolpert (KW) variance is based on the bias-variance decomposition of the error of a classifier (see Kohavi et al. [1996]). On an instance x, the variability of the predicted class label y is defined as

$$var_x = \frac{1}{2}\Big(1 - \sum_{y \in \{-1,+1\}} P^2(y|x)\Big)$$

Kuncheva et al. [2003] modified the variability to measure diversity by considering two classifier outputs: correct (denoted by $\tilde{y} = +1$) and incorrect (denoted by $\tilde{y} = -1$). They estimated $P(\tilde{y} = +1|x)$ and $P(\tilde{y} = -1|x)$ over individual classifiers as follows:

$$P(\tilde{y} = +1|x) = \frac{\rho(x)}{N} \text{ and } P(\tilde{y} = -1|x) = 1 - \frac{\rho(x)}{N}$$

where $\rho(x)$ is the number of individual classifiers that classify x correctly. By substituting these formulas in the KW-variance and averaging over the data set D the $\kappa\omega$ measure is obtained

$$\kappa\omega = \frac{1}{mN^2} \sum_{k=1}^{m} \rho(x_k)\big(N - \rho(x_k)\big)$$

where the larger the $\kappa\omega$ measurement, the larger the diversity.
Interrater agreement is a measure of interrater (interclassifier) reliability (see Fleiss [1981]). It can be used to measure the level of agreement within a set of classifiers (see Kuncheva et al. [2003]). It is measured as follows

$$\kappa = 1 - \frac{\frac{1}{N}\sum_{k=1}^{m} \rho(x_k)\big(N - \rho(x_k)\big)}{m(N-1)\overline{p}(1-\overline{p})}$$

where $\rho(x_k)$ is the number of classifiers that classify x_k correctly, and

$$\overline{p} = \frac{1}{mN} \sum_{i=1}^{N} \sum_{k=1}^{m} I(h_i(x_k) = y_k)$$

is the average accuracy of individual classifiers. Again, $\kappa = 1$ if the classifiers totally agree on D, and $\kappa \leqslant 0$ if the agreement is even less than what is expected by chance.
Kuncheva et al. [2003] showed that the KW-variance, the averaged disagreement dis_{avg} and the kappa-statistic κ were closely related. The relations are as follows

$$\kappa\omega = \frac{(N-1)}{2N} dis_{avg} \text{ and } \kappa = 1 - \frac{N}{(N-1)\overline{p}(1-\overline{p})}\kappa\omega$$

where $\overline{p}$ is as above.
In entropy, for an instance x_k, the disagreement will be maximised if a tie occurs in the votes of individual classifiers. Cunningham et al. [2000] directly calculated the Shannon's entropy on every instance and averaged them over D for measuring diversity, getting

$$Ent_{cc} = \frac{1}{m} \sum_{k=1}^{m} \sum_{y \in \{-1,+1\}} -P(y|x_k) \log P(y|x_k)$$

where

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$P(y|x_k) = \frac{1}{N}\sum_{i=1}^{N} I(h_i(x_k) = y)$$

can be estimated by the proportion of individual classifiers that predict y as the label of x_k. The computation of Ent_{cc} does not require to know the correctness of individual classifiers.
Assuming the correctness of the classifiers known, Shipp et al. [2002] defined their entropy measure as

$$Ent_{sk} = \frac{1}{m}\sum_{k=1}^{m} \frac{\min\big(\rho(x_k), N - \rho(x_k)\big)}{N - \lceil \frac{N}{2} \rceil}$$

where $\rho(x_k)$ is the number of individual classifiers that classify x correctly. The value of Ent_{sk} is in the range of $[0, 1]$, where 0 indicates no diversity and 1 indicates the largest diversity. Note, even though this formula does not use the logarithm function, it is easier to handle and faster to calculate.
Assuming that the diversity is maximised when the failure of one classifier is accompanied by the correct prediction of the other, Partridge et al. [1997] defined the generalised diversity as follows

$$gd = 1 - \frac{p(2)}{p(1)}$$

where

$$p(1) = \sum_{i=1}^{N} \frac{1}{N} p_i \text{ and } p(2) = \sum_{i=1}^{N} \frac{1}{N}\frac{(i-1)}{(N-1)} p_i$$

where p_i denotes the probability of i randomly chosen classifiers failing on a randomly drawn instance x. The gd value is in the range of $[0, 1]$, and the diversity is minimised when $gd = 0$. The authors also proposed a modified version of the generalised diversity, called coincident failure (CF), given by

$$cfd = \begin{cases} 0 \text{ if } p_0 = 1 \\ \frac{1}{1-p_0}\sum_{i=1}^{N} \frac{(N-i)}{(N-1)} p_i \text{ if } p_0 < 1 \end{cases}$$

In that setting, $cfd = 0$ if all classifiers give the same predictions simultaneously, and $cfd = 1$ if each classifier makes mistakes on unique instances.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Part II

Neural Networks

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 5

Introduction to artificial neural networks

While wavelet analysis enable to decompose complex systems into simpler elements (see Appendix (23.3)), in order to understand them, we can also gather simple elements to produce a complex system. Networks is one approach among others achieving that goal. They are characterised by a set of interconnected nodes seen as computational units receiving inputs and processing them to obtain an output. The connections between nodes, which can be unidirectional or bidirectional, determine the information flow between them. We obtain a global behaviour of the network, called emergent, since the abilities of the network supercede the one of its elements, making networks a very powerful tool. Since Neural Nets have been widely studied by computer scientists, electronic engineers, biologists, psychologists etc, they have been given many different names such as Artificial Neural Networks (ANNs), Connectionism or Connectionist Models, Multi-layer Percepetrons (MLPs), or Parallel Distributed Processing (PDP) to name a few. However, a small group of classic networks emerged as dominant such as Hopfield Networks (see Hopfield [1982]), Back Propagation (see Rumelhart et al. [1985] [1986b]), Competitive Networks and networks using Spiky Neurons.

Among the different networks existing, the artificial neural networks (ANNs) and the artificial recurrent neural networks (RNNs) are computational models designed by more or less detailed analogy with biological brain modules. They are inspired from natural neurons receiving signals through synapses located on the dendrites, or membrane of the neuron. When the signals received are strong enough (surpass a certain threshold), the neuron is activated and emits a signal through the axon, which might be sent to another synapse, and might activate other neurons. ANN is a high level abstraction of real neurons consisting of inputs (like synapses) multiplied by weights (strength of the respective signals), and computed by a mathematical function determining the activation of the neuron. Another function computes the output of the artificial neuron (sometimes in dependence of a certain threshold). Depending on the weights (which can be negative), the computation of the neuron will be different, such that by adjusting the weights of an artificial neuron we can obtain the output we want for specific inputs. In presence of a large number of neurons we must rely on an algorithm to adjust the weights of the ANN to get the desired output from the network. This process of adjusting the weights is called learning or training. The aim of training the network is to obtain the optimal network minimising the difference between the output and a target, given some input data. Then, the output of the optimal network becomes the optimal predictor. The optimisation process is usually done by gradient descent method, where the weights are updated by an amount being equal to the opposite of the gradient. This way of updating weights is called the standard back propagation.

See textbooks by Fausett [1994], Haykin [2005], Bishop [2006], Mohri et al. [2012], Goodfellow et al. [2016], among others.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

5.1 The classic networks

Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples. The basic architecture of an ANN is a network comprised of nodes (called *neurons* in biology) interacting with each other via incoming and outgoing weighted connections indicating the strength degree of dendrite between neurons. While McCulloch et al. [1943] introduced the first ANN model, most modelling of neural learning networks has been based on synapses of a general type described by Hebb [1949] and Eccles [1953]. Given some input, the network will be activated and this activation signal will be passed throughout the rest of the network through the connections. Generally, an ANN model is specified by its topology, node characteristics and the training algorithms. There are a variety of ANNs with various topologies, node properties and training algorithms. In order to make a neural network model which has the same capability for pattern recognition as a human being, Rosenblatt [1957] [1958] introduced the principles of perceptrons. Perceptrons were mainly modelled with neural connections in a forward direction $A \to B \to C \to D$. Consequently, the analysis of networks with strong backward coupling $A \leftrightarrows B \leftrightarrows C$ proved intractable. Further, perceptron modelling required synchronous neurons like a conventional digital computer. Rosenblatt [1962] proposed the theory of three-layer series-coupled perceptrons. He also started the theory of multi-layer and cross-coupled perceptrons. Ivakhnenko [1971] described a multilayered perceptronlike network structure with 8 layers trained by the group method of data handling algorithm. Hopfield [1982] overcame the difficulties around the perceptron and triggered the research interest in the area by demonstrating the computational capabilities of networks with symmetric connections. The main contribution of the paper was the introduction of the energy function of the network which turns to be their most useful property in the context of optimisation. To improve on the limitation of the perceptron, Fukushima [1980] proposed the multilayer percetron (MLP) also called feedforward network. Two of the most important distinct types of ANNs are feedforward and feedback network. ANNs with cycles are *feedback* networks, which are also referred to as *recurrent* neural networks (RNNs). Those networks with acyclic connections are called *feedforward* neural networks (FNNs). Algorithms such as back-propagation use gradient descent to tune network parameters to best fit a training set of input-output pairs, making the learning process robust to errors in the training data (see Rumelhart et al. [1985] [1986b]). Various successful applications to practical problems developed, such as learning to recognise handwritten characters (see LeCun et al. [1989]) and spoken words (see Lang [1990]), learning to detect fraudulent use of credit cards, drive autonomous vehicles on public highways (see Pomerleau [1993]). Rumelhart et al. [1994] provided a survey of practical applications. Since McCulloch et al. [1943] introduced the first ANN model, various models where developed with different functions, accepted values, topology, learning algorithms, hybrid models where each neuron has a larger set of properties etc. While there is various types of classifiers, neural network based classifiers dominate the literature. Yet, compared to traditional NNs, higher order neural networks (HONNs) have several unique characteristics, including

- stronger approximation with faster convergence property
- greater storage capacity
- higher fault tolerance capability

However, its major drawback is the exponential growth in the number of weights with the increasing order of the network. As a special case of the feedforward HONN, the Pi-Sigma networks (PSNs) introduced by Shin et al. [1991] have the capability of higher order neural networks, but using a smaller number of weights. In order to enhance the learning capability of neural network, many researchers have improved the system by combining other techniques such as fuzzy logic (see Zadeh [1994]), genetic algorithm (see Harrald et al. [1997]).

For simplicity of exposition we are first going to introduce the perceptron and present the Hopfield networks (see Hopfield [1982]). We will then present the multilayer percetron (MLP), and then briefly describe the back-propagation algorithm proposed by Rumelhart et al. [1985].

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

5.1.1 The perceptron

Rosenblatt [1957] presented the perceptron algorithm which was intended to be a machine. Even though its first implementation was in software for the IBM 704, it was subsequently implemented in custom-built hardware as the Mark 1 perceptron. This machine was designed for image recognition: it had an array of 400 photocells, randomly connected to the neurons. Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors.
The perceptron is an algorithm for learning a binary classifier (see Section (4.1)) called a threshold function: a function that maps its input $x \in \mathbb{R}^n$ (a real-valued vector) to an output value $f(x)$

$$f(x) = \begin{cases} 1 \text{ if } w \cdot x + b > 0 \\ 0 \text{ otherwise} \end{cases}$$

where w is a vector of real-valued weights, $w \cdot x$ is the dot product $\sum_{i=1}^{n} w_i x_i$, where n is the number of inputs to the perceptron, and b is the bias. The bias shifts the decision boundary away from the origin and does not depend on any input value. Figure (5.1) illustrates the mechanism of a perceptron.

Figure 5.1: A perceptron as a cell body

The value of the activation function $f(x)$ is used to classify x as either a positive or a negative instance, in the case of a binary classification problem. Spatially, the bias alters the position (though not the orientation) of the decision boundary. The perceptron learning algorithm does not terminate if the learning set is not linearly separable. If the vectors are not linearly separable learning will never reach a point where all vectors are classified properly. Minsky et al. [1969] documented the limitations of the perceptron. The most famous example of the perceptron's inability to solve problems with linearly nonseparable vectors is the Boolean exclusive-or problem (XOR) (see Section (2.1)).
Although the perceptron initially seemed promising, it was quickly proved that perceptrons could not be trained to recognise many classes of patterns. This is because simple two-layer associative networks can only map similar input patterns to similar output patterns. As a result, such systems were unable to learn certain mappings from input to output. This caused the field of neural network research to stagnate for many years, before it was recognised that a feedforward neural network with two or more layers (also called a multilayer perceptron) had far greater processing power than perceptrons with one layer (also called a single layer perceptron).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

5.1.2 The Hopfield networks

5.1.2.1 Description

The Hopfield neural network model (see Hopfield [1982]) consists of a fully connected network of units (or neurons). The connections between the units are weighted, such that for any two units i and j, w_{ij} is the weight of the connection between them. The model assumes symmetrical weights ($w_{ij} = w_{ji}$) and in most of the cases zero self-coupling terms ($w_{ii} = 0$). A connection with positive weight is an excitatory connection, while a connection with negative weight is an inhibitory connection. A unit i is characterised by its output (or state) v_i, the activation (or network input) u_i that receives from the other units, and a threshold θ_i. The network state is given by the output (or state) vector $v = \big(v_1, v_2, ..., v_n\big)$.
Each unit receives input from all the other units and forwards its output to all the other units. The output of a unit is updated by the dynamics of the network. In general, the output behaviour is described as a function of the activation, where the activation depends on the weighted summation of the inputs and the threshold. In general, we follow the dynamics rule from McCulloch et al. [1943], given by

$$v_i(t+1) = \Phi(u_i(t)) = \Phi\big(\sum_{j=1}^{n} w_{ij} v_j(t) + \theta_i\big) \tag{5.1.1}$$

where the activation is

$$u_i(t) = w_i \cdot v(t) + \theta_i = \sum_{j=1}^{n} w_{ij} v_j(t) + \theta_i \tag{5.1.2}$$

and θ_i is the threshold. The activation function $\Phi(x)$ can take several forms:

- it can be the unit step function (Heaviside function)

$$\Theta(x) = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ if } x \leqslant 0 \end{cases}$$

  in which case we have a discrete Hopfield network with binary states $\{0, 1\}$.

- the sign function

$$\text{sgn}(x) = \begin{cases} 1 \text{ if } x \geqslant 0 \\ -1 \text{ if } x < 0 \end{cases}$$

  in which case the network is discrete but with values $\{1, -1\}$.

- a network with continuous-valued units (analog Hopfield network), where the values fall in the range $[0, 1]$ or $[-1, 1]$ can be obtained with the sigmoid function.

  - In the case of the scaled logistic function we get

$$g_\beta(x) = \frac{1}{1 + e^{-2\beta x}}$$

    where $\lim_{x\to\infty} g_\beta(x) = 1$ and $\lim_{x\to-\infty} g_\beta(x) = 0$. Its graph is s-shaped, it is differentiable and monotonic. The parameter β is usually taken as $\beta = \frac{1}{T}$ where T is a virtual temperature. The latter adjusts the sharpness of the sigmoid (or hyperbolic tangent) function and at the limit $T \to 0$ (absolute temperature) the output becomes discrete.

  - We can also use the hyperbolic tangent function

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

    which can be regarded as linear transformation of the logistic sigmoid onto the interval $[-1, 1]$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

– The generalised logistic function is given by

$$f(x) = (1 + e^{-x})^{-\alpha} \, , \alpha > 0$$

Note that

$$\text{sgn}(x) = 2\Theta(x) - 1 \text{ and } tanh(\beta x) = 2g_\beta(x) - 1$$

so that the $\{0, 1\}$ (or $[0, 1]$) model can be easily transformed to $\{-1, 1\}$ (or $[-1, 1]$) and vice-versa. In the limit $\beta \to \infty$ the values become discrete.
All the networks above are deterministic in the sense that the next state is an explicit function of the previous state and the characteristics of the network. In the case of the stochastic Hopfield networks the probability of a unit to be in a particular state is drawn from a probability distribution such as Boltzmann or Cauchy. Alternatively, the stochastic network can be viewed as a deterministic network, where the threshold of a unit is variable and is drawn from a probability density.
Note, the updating policy can be

- synchronous, in which case all the units are updated simultaneously at each time step, or
- asynchronous, where either the units are updated in sequence one at each time step (sequential asynchronous update) or at each time step one randomly chosen unit is updated (random asynchronous update).

In summary, the properties that characterise different Hopfield networks are

- Discrete vs Continuous-Valued Units
- Deterministic vs Stochastic Networks
- Synchronous vs Asynchronous Update (Sequential or Random)

5.1.2.2 The energy function

The essential ingredient of the learning process is the Hebbian property which is the modification of w_{ij} by correlations like

$$\Delta w_{ij} = \left[v_i(t)v_j(t)\right]_{average}$$

where the average is some appropriate calculation over past history. In addition, the Hopfield model has stable limit points. That is, the energy function of a Hopfield network is a function defined over the state space of the network given by

$$\begin{aligned} E(v) &= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} v_i v_j w_{ij} - \sum_{i=1}^{n} v_i \theta_i \\ &= -\sum_{i=1}^{n} v_i \Big(\frac{1}{2}\sum_{j=1}^{n} v_j w_{ij} + \theta_i\Big) \\ &= -\sum_{i=1}^{n} v_i u_i = -u \cdot v \end{aligned} \tag{5.1.3}$$

Note, this function always decreases (not necessarily monotonically because of the threshold) during the evolution of the network due to the dynamics of the systems given in Equation (5.1.1). State changes (altering v_i) will continue until a least (local) E is reached. This case is isomorphic with an Ising model.
Thus, the energy function can be viewed as defining an energy landscape and the dynamics can be thought of as the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

motion of a small sphere on the energy surface under the influence of gravity and friction. Consequently, the network performs as a minimiser of the energy function and will reach a local or global minimum of the function during its evolution.
One can show that whenever a unit i changes state, the energy difference ΔE_i is given by

$$\Delta E_i = -u_i \Delta v_i = -\big(\sum_{j=1}^{n} w_{ij} v_j + \theta_i\big)\Delta v_i \tag{5.1.4}$$

where u_i is the activation in Equation (5.1.2). As an example, we consider the $\{0, 1\}$ network with dynamics given in Equation (5.1.1). Then

- If the activation is positive, then either $\Delta u_i = 0$ $(1 \to 1)$ or $\Delta u_i = 1$ $(0 \to 1)$ and thus $\Delta E_i \leqslant 0$.
- If the activation is negative, then either $\Delta u_i = 0$ $(0 \to 0)$ or $\Delta u_i = -1$ $(1 \to 0)$ and thus $\Delta E_i \leqslant 0$.

The argument is similar for the other models. Note, the main property of the energy function given in Equation (5.1.3) is held only if the weights of the connections are symmetric and the self-coupling terms are zero or positive. This is the case for the Hopfield network so that its energy function is called Lyapunov function.

5.1.3 The multilayer perceptron

5.1.3.1 Description

We have seen above that to understand the capability of higher organisms for perceptual recognition, generalisation, recall, and thinking, several brain models were proposed, which amount to logical contrivances for performing particular algorithms in response to sequences of stimuli. In order to characterise the gross organisation in biological systems, these models have been formulated in terms of probability theory rather than symbolic logic (see Rosenblatt [1958]). This approach led to the development of simple two-layer associative networks in which a set of input patterns arriving at an input layer are mapped directly to a set of output patterns at an output layer. They map similar input patterns to similar output patterns. However, they are unable to learn certain mappings from input to output due to the lack of hidden units. Minsky et al. [1969] analysed the conditions under which such systems are capable of carrying out the required mappings. They showed that if we have the right connections from the input units to a large enough set of hidden units, we can always find a representation that will perform any mapping from input to output through these hidden units. This is illustrated in Figure (5.2).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 5.2: Multilayer network

The difficulty becomes that of obtaining a guaranteed learning rule for all problems that can be solved in networks with hidden units. Several solutions have been proposed such as employing simple unsupervised learning rules so that useful hidden units develop. Another solution is to assume an internal representation that, on some a priori grounds, seems reasonable. Alternatively, one can attempt to develop a learning procedure capable of learning an internal representation adequate for performing the task at hand. This procedure involves the use of stochastic units (see Boltzmann machines).

In a different setting, Fukushima [1980] proposed a self-organised network by learning without a teacher for visual pattern recognition called neocognitron. The network is a multilayer structure consisting of an input layer (photoreceptor array) followed by a cascade connection of a number of modular structures, each of which is composed of two layers of cells connected in a cascade. This is illustrated in Figure (5.3).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 5.3: Interconnections between layers in the neocognitron

In order to obtain a guaranteed learning rule for all problems that can be solved in networks with hidden units the backpropagation algorithm became the most efficient learning procedure. It is essentially a judicious application of the chain rule and gradient descent. There are several ways for deriving backpropagation. For instance, Parker [1985] proposed generalised delta rule called the learning logic. LeCun [1985] [1986] proposed to use local criteria attached to each unit which are minimised locally. Different constraints can be assigned to these local criteria. However, Rumelhart et al. [1985] [1986b] presented the simplest one by elaborating on the generalised delta rule and provided results on well known problems (see details in Section (5.1.4)). It works with deterministic units and involves only local computations. Given a set of pairs of input and output patterns, the system first uses the input vector to produce its own output vector and then compares this with the desired output, or target vector. If there is no difference, no learning takes place. Otherwise the weights are changed to reduce the difference. For linear units it minimises the squares of the differences between the actual and the desired output values summed over the output units and all pairs of input/ output vectors. In the special case of no hidden units, we recover the standard delta rule. This algorithm is a layer feed-forward ANN, since the artificial neurons are organised in layers with signals sent forward and with errors propagated backwards. The network receives inputs via neurons in the input layer, and the output of the network is given via neurons on the output layers. There may be one or more intermediate hidden layers. The authors showed that the error propagation scheme leads to solutions in virtually every case. LeCun [1988] presented a mathematical framework for studying backpropagation based on the Lagrangian formalism. The backpropagation is formulated as an optimisation problem with non-linear constraints (see Section (18.5)). It provides a simple formulation of continuous recurrent network equations (see Pineda [1987]).

The approach proposed by Rumelhart et al. [1985] [1986b] led to the popular *multilayer perceptron* (MLP) which is formed in layers, the idea being that multilayer ANN can approximate any continuous function (see Section (5.1.3.3)). Note, the MLP is also called feedforward network. The term MLP is loosely used to refer to any feedforward ANN, or sometimes strictly to refer to networks composed of multiple layers of perceptrons. For simplicity of exposition, we are going to consider a network with three layers, that is, an input layer, a hidden layer, and an output layer. The hidden layer is used for capturing the non-linear relationships among variables. It is illustrated in Figure 5.4, where the bottom layer is the *input layer*. Then the information is propagated to the middle layer, called the *hidden layer*. Finally, the output layer receives the incoming value. This process is termed the *forward pass*. Note that the S-shaped curve is the squashing function which forces the output of the unit to fall between certain bounded interval. Except for the input layer, the output value of each unit in the other layers should be passed through the squashing function before transferring to the next layer. The back-propagation algorithm uses supervised learning where we provide the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

algorithm with examples of the inputs and outputs we want the network to compute, and then the error (difference between actual and expected results) is calculated. Defining the network function f_{out} as a particular implementation of a composite function from input to output space, the learning problem consists in finding the optimal combination of weights so that f_{out} approximates a given function f as closely as possible (see Section (10.1.1)). However, in practice the function f is not given explicitly but only implicitly through some examples.

Figure 5.4: **A multilayer perceptron.** The three layers are input, hidden, and output layers, respectively, from bottom to top. The S-shaped curves in the hidden and output layers are the sigmoid squashing functions applying to the net values.

MLPs were a popular machine learning solution in the 1980s, finding applications in diverse fields such as speech recognition, image recognition, and machine translation software. Interest in backpropagation networks returned with the increasing successes of deep learning.

5.1.3.2 A mathematical representation

We extend the mathematical representation of the Hopfield network in Section (5.1.2) to the multilayer perceptron. We let $L \in \mathbb{N}$ denotes the number of layers (depth) and the tuple $(N_1, N_2, ..., N_L)$ be the number of nodes (neurons) on each layer. We consider the affine function

$$\begin{aligned} U^l : \mathbb{R}^{N_l} &\rightarrow \mathbb{R}^{N_{l+1}} \text{ for } 1 \leqslant l \leqslant L-1 \\ x &\rightarrow W^l x + b^l \end{aligned}$$

acting between layers for some matrix $W^l \in \mathbb{R}^{N_{l+1} \times N_l}$. In that setting, $x \in \mathbb{R}^{N_l}$ is the input vector, the vector $b^l \in \mathbb{R}^{N_{l+1}}$ is the bias term, and each entry $W^l_{(j,i)}$ denotes the weight connecting node $i \in N_l$ of layer l to the node $j \in N_{l+1}$ of layer $l+1$. Note, the weight matrix can also be denoted $W^l \in \mathbb{R}^{N_l \times N_{l+1}}$ where each entry $W^l_{(i,j)}$ denotes the weight connecting node $i \in N_l$ of layer l to the node $j \in N_{l+1}$ of layer $l+1$. In that case, the affine function is $U^l : x \rightarrow (W^l)^\top x + b^l$, where $(W^l)^\top$ is the transpose matrix. This is illustrated in Figure (5.5).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 5.5: Affine function for going from layer l with N_l input to layer $l+1$ with N_{l+1} output.

In order to represent the affine functions on each layer we let the tuple $U = (U^1, ..., U^L)$ be the network weights for any such collection of affine functions. Then, the neural network $F(U, \cdot) : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ is defined as the composition

$$F = F_L \circ \cdots \circ F_1$$

where each component (or layer) is of the form

$$F_l = \Phi^l \circ U^l \tag{5.1.5}$$

where the function $\Phi^l : \mathbb{R} \to \mathbb{R}$ is the activation function. That is, the lth layer F_l is an affine transform followed by a non-linear activation function.

***Remark* 5.1.1** *If a multilayer perceptron has a linear activation function Φ^l in all neurons, that is, a linear function that maps the weighted inputs to the output of each neuron, then linear algebra shows that any number of layers can be reduced to a two-layer input-output model.*

In our setting, the first and last layers, F_1 and F_L, are respectively the input and output layers. The layers in between, $F_2, ..., F_{L-1}$, are the hidden layers. The set of affine functions $U^l : \mathbb{R}^{N_l} \to \mathbb{R}^{N_{l+1}}$, for $1 \leqslant l \leqslant L-1$ is characterised by the set of parameters $U \in \mathbb{R}^{\sum_{i=1}^{L-1} N_i(1+N_{i+1})} = \Omega$ so that the number of network weights to calibrate is $\sum_{i=1}^{L-1} N_i(1+N_{i+1})$.

Considering a single neurone, for any $j \in N_{l+1}$ and any input vector $x \in \mathbb{R}^{N_l}$, the output of the node j of layer $l+1$ is

$$x \to \Phi^l\big(w_j^\top x + b_j^l\big)$$

where the input vector x is given by the output of all nodes in the previous layer, $b_j^l \in \mathbb{R}$ is the bias, and the weight vector $w_j = W^l_{(j,\cdot)} \in \mathbb{R}^{N_l}$. It is the set of weights connecting node j to the previous layer. Note, we recover the perceptron in Section (5.1.1).
In the special case of a feedforward neural network (FNN) with one layer of sigmoidal nonlinearities, we get the input vector $x \in N_1$, the weight vector $w_i \in \mathbb{R}^{N_1}$, where $i = 1, ..., N_2$, and the network is defined by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$F_2(x) = \sum_{i=1}^{N_2} v_i \Phi^1(w_i^\top x + b_i)$$

where $\Phi^1(\cdot)$ is a sigmoidal function, and $v_i \in \mathbb{R}$ is some weight.

5.1.3.3 The universal approximation theorem

Cybenko [1989] hinted at the universal approximation theorem for sigmoid activation functions. Funahashi [1989] and Hornik et al. [1989] showed that a MLP with one hidden layer and sufficient non-linear units could approximate any continuous function on a compact input domain to arbitrary precision. Therefore, MLP are recognised as universal approximators. Hornik [1991] showed that it is not the specific choice of the activation function, but rather the multilayer feedforward architecture itself which gives neural networks the potential of being universal approximators. The universal approximation theorem can be stated as follows:

Theorem 5.1.1 *The universal approximation theorem*
Let $\Phi(\cdot)$ be a nonconstant, bounded, and continuous function. Let I_m denotes the m-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\epsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$, where $i = 1, ..., N$, such that we may define:

$$F_2(x) = \sum_{i=1}^{N} v_i \Phi(w_i^\top x + b_i) \tag{5.1.6}$$

as an approximate realisation of the function f where f is independent of Φ. That is,

$$|F_2(x) - f(x)| < \epsilon$$

for all $x \in I_m$. In other words, functions of the form $F_2(x)$ are dense in $C(I_m)$.

Note, the theorem still holds when replacing I_m with any compact subset of $\mathbb{R}^m$.
Hornik et al. [1990] gave conditions ensuring that multilayer feedforward networks (FFNs) with as few as a single hidden layer and an appropriately smooth hidden layer activation function were capable of arbitrarily accurate approximation to an arbitrary function and its derivatives. Note, these networks only need to possess a generalised derivative, as in the case for certain piecewise differentiable functions (see Appendix (23.2.1.2)). They consider the class of single hidden layer FFNs having output functions belonging to the set

$$\Sigma(\Phi) \equiv \{F_2 : \mathbb{R}^r \to \mathbb{R} \mid F_2(x) = \sum_{j=1}^{q} v_j \Phi(w_j^\top x + b_j)\}$$

where $x \in \mathbb{R}^r$, $r \in \mathbb{N}$, is a vector of input, w_j is the input to hidden layer weights, $v_j \in \mathbb{R}$ is the hidden to output layer weights, $q \in \mathbb{N}$ is the number of hidden units, and Φ is a given hidden unit activation function. The first partial derivatives of the network output function are given by

$$\frac{\partial F_2(x)}{\partial x_i} = \sum_{j=1}^{q} v_j w_{ji} D\Phi(w_j^\top x + b_j)\,, i = 1, ..., r$$

where x_i is the ith component of x, w_{ji} is the ith component of w_j and $D\Phi$ denotes the first derivative of Φ. We let $S_p^m(U,\mu)$ be the first Sobolev space defined as the collection of all functions f in $C^m(U)$ such that $\|D^\alpha f\|_{p,U,\mu} < \infty$ for all $|\alpha| \leqslant m$. We also let $C_\downarrow^\infty(\mathbb{R}^r)$ be the space of rapidly decreasing function in $C^\infty(\mathbb{R}^r)$. The main result is as follows:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 5.1.2 *Let $\Phi \neq 0$ belongs to $S_1^m(\mathbb{R}, \lambda)$ for some integer $m \geqslant 0$. Then, $\Sigma(\Phi)$ is m-uniformly dense on compacta in $C_{\downarrow}^{\infty}(\mathbb{R}^r)$.*

Thus, as long as the hidden layer activation function Φ belongs to $S_1^m(\mathbb{R}, \lambda)$ and does not vanish everywhere, then $\Sigma(\Phi)$ can approximate any function belonging to $C_{\downarrow}^{\infty}(\mathbb{R}^r)$ and its derivatives up to order m arbitrarily well on compact sets.
Kurkova [1992] took advantage of techniques developed by Kolmogorov to give a direct proof of the universal approximation capabilities of perceptron type networks with two hidden layers. Telgarsky [2016] argued that the theorem was only true in the case where the hidden layers were exponentially large.

5.1.4 Gradient descent and the delta rule

5.1.4.1 The gradient descent rule

As explained by Rumelhart et al. [1985] [1986b], one way forward to finding the optimal combination of weights in the multilayer perceptron is to consider the delta rule, which uses gradient descent to search the hypothesis space of possible weight vectors to find the weights best fitting the training examples. The gradient descent provides the basis for the backpropagation algorithm, which can learn networks with many interconnected units.
Given a vector of input $x \in \mathbb{R}^n$ together with a weight vector $w \in \mathbb{R}^n$, we consider the task of training an unthresholded perceptron, that is, a linear unit for which the output O is given by

$$O(x) = w \cdot x = \sum_{i=1}^{n} w_i x_i$$

Note, a linear unit is the first stage of a perceptron without the threshold (see Section (5.1.1)). We are also given a training set $\{(x_1, d_1), ..., (x_P, d_P)\}$ consisting of P ordered pairs of n- and 1-dimensional vectors called the input and output patterns, respectively. We must specify a measure for the training error of a hypothesis (weight vector), relative to the set $\mathcal{D}$ of training examples (see Section (10.1)). A common measure is to use the mean squared error (MSE)

$$E(w) = \frac{1}{2} \sum_{p=1}^{P} (O_p - d_p)^2$$

where E is the error function, d_p is the target output for training example p, and O_p is the output of the linear unit for training example p. One can show that under certain conditions, the hypothesis minimising the error function E is also the most probable hypothesis in H given the training data. The entire hypothesis space of possible weight vectors and their associated E values produce an error surface. Given the way in which we defined E, for linear units, this error surface must always be parabolic with a single global minimum.
The gradient descent search (see Section (18.3.1.1)) determines a weight vector minimising the error function E by starting with an arbitrary initial weight vector, and then repeatedly modifying it in small steps. At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface, until a global minimum error is reached. This direction is found by computing the derivative of E with respect to each component of the vector w, called the gradient of the error function E with respect to w, given by

$$\nabla E(w) = [\frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n}]$$

The gradient specifies the direction that produces the steepest increase in the error function E, and its negative produces the steepest decrease. Hence, the training rule for gradient descent is

$$w \longleftarrow w + \Delta w$$

where

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\Delta w = -\eta \nabla E(w)$$

where $\eta > 0$ is the learning rate determining the step size in the gradient descent search. The negative sign implies that we move the weight vector in the direction that decreases the function E. Given the definition of $E(w)$, we can easily differentiate the vector of $\frac{\partial E}{\partial w_i}$ derivatives as

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{p=1}^{P} (O_p - d_p)^2 \text{ , } i = 1, ..., n$$

which gives after some calculation

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial E}{\partial O_p} \frac{\partial O_p}{\partial w_i} \\ &= \sum_{p=1}^{P} (O_p - d_p) \frac{\partial O_p}{\partial w_i} \\ &= \sum_{p=1}^{P} (O_p - d_p) x_{i,p} \end{aligned}$$

where $x_{i,p}$ is the single input component x_i for training example p. The weight update rule for gradient descent is

$$\Delta w_i = -\eta \sum_{p=1}^{P} (O_p - d_p) x_{i,p} = \eta \sum_{p=1}^{P} (d_p - O_p) x_{i,p}$$

In the case where η is too large, the gradient descent search may overstep the minimum in the error surface rather than settling into it. One solution is to gradually reduce the value of η as the number of gradient descent steps grows. Gradient descent is a strategy for searching through a large or infinite hypothesis space which can be applied whenever

1. the hypothesis space contains continuously parameterised hypotheses.
2. the error can be differentiated with respect to these hypothesis parameters.

The key practical difficulties in applying gradient descent are:

1. Converging to a local minimum can sometimes be quite slow.
2. If there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

5.1.4.2 The delta rule

While the gradient descent training rule computes weight updates after summing over all the training examples in $\mathcal{D}$, the stochastic gradient descent (SGD) approximates this gradient descent search by updating weights incrementally, following the calculation of the error for each individual example. Hence, as we iterate through each training example, we update the weight according to

$$\begin{aligned} \Delta w_i &= \eta (d_p - O_p) \frac{\partial O_p}{\partial w_i} \\ &= \eta (d_p - O_p) x_i \end{aligned} \tag{5.1.7}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where the subscript i represents the ith element for the training example in question. One way of viewing this stochastic gradient descent is to consider a distinct error function $E_p(w)$ defined for each individual training example p as follow

$$E_p(w) = \frac{1}{2}(d_p - O_p)^2 \tag{5.1.8}$$

where d_p and O_p are the target value and the unit output value for training example p. We therefore iterates over the training examples p in $\mathcal{D}$, at each iteration altering the weights according to the gradient with respect to $E_p(w)$. The sequence of these weight updates provides a reasonable approximation to descending the gradient with respect to the original error function $E(w)$. This training rule is known as the Widrow-Hoff rule (see Widrow et al. [1960]), or the delta rule, or sometimes the least-mean-square (LMS) rule. Widrow et al. devised the ADALINE (Adaptive Linear Neuron or later Adaptive Linear Element), which is the name of a physical device that implemented a single layer artificial neural network. In the learning phase, the weights are adjusted according to the weighted sum of the inputs. The difference

$$(d_p - O_p) \tag{5.1.9}$$

in Equation (5.1.7) represents the scalar error between the prediction $O_p = w \cdot x$ and the target value d_p. It is multiplied by the observation x_i to determine the weight changes because x_i indicates how the changing weight will affect the error. Note, the delta rule converges only asymptotically toward the minimum error hypothesis, but it converges regardless of whether the training data are linearly separable.

5.1.5 Accounting for time and earning profits

In traditional time series forecasting, the criteria for assessing model performance are error functions based on the goodness of fit of target and predicted values. Since neural networks are similar to conventional regression estimators, except for their nonlinearity, error functions are also used to judge the goodness of fit of the model. However, normalised mean square error (NMSE) and other error functions are not the most appropriate measures in finance.

5.1.5.1 Time factor

For instance, weighting all data equally (ordinary leasy squares) are less accurate than discounted least squares, which weight the most recent data more heavily (see Makridakis et al. [1982]). Incorporating time factor to neural networks, Refenes et al. [1997] proposed the discounted least squares (DLS) neural network model. Assuming a feed-forward network with n input and a single output unit ($m = 1$) with k hidden layers, we let the ordinary least-squares criterion of the network be given by

$$E_{LS} = \frac{1}{2P}\sum_{p=1}^{P}(O_p - d_p)^2$$

where O_p and d_p denote the pth output and target value, respectively.

***Remark* 5.1.2** *In general, when considering a training set consisting of sequential data, or time-varying input, we represent them as $\{(\overline{x}_1, \overline{d}_1), ..., (\overline{x}_T, \overline{d}_T)\}$ with T ordered pairs. To be consistent with the articles described we keep the P ordered pairs, and we let $p = P$ be the most recent observation.*

Refenes et al. [1997] proposed a simple modification to the error backpropagation procedure taking into account gradually changing input-output pairs. The procedure is based on the principle of discounted least-squares whereby learning is biased towards more recent observations with long term effects experiencing exponential decay through time. The cumulative error calculated by the DLS procedure is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$E_{DLS} = \frac{1}{2P}\sum_{p=1}^{P}\phi(p)(O_p - d_p)^2$$

where $\phi(p)$ is an adjustment of the contribution of observation p to the overall error. They chose the simple sigmoidal decay function

$$\phi_{DLS}(p) = \frac{1}{1+e^{a-bp}}$$

where $b = \frac{2a}{P}$ and a is the discount rate. The learning rule is derived in the usual way by repeatedly changing the weights by an amount proportional to

$$\frac{\partial E_{DLS}}{\partial W} = \phi_{DLS}(p)\frac{\partial E_{LS}}{\partial W}$$

since the discount factor $\phi(p)$ is a function of the recency of the observation which is independent of the actual order of pattern presentation within the learning process. Hence, the algorithm is not affected by randomising the order in which patterns are presented and can be applied to both batch and stochastic update rules. Using a sine wave with changing amplitude and a non-trivial application in exposure analysis of stock returns to multiple factors, Refenes et al. [1997] compared the performance of the two cost functions on a backpropagation network with a single hidden layer of 12 units and batch update. They showed that DLS was a more efficient procedure for weakly non-stationary data series.

5.1.5.2 Direction measures

Further, we are usualy more interested in earning profits than in the quality of the forecast itself. When classifying returns, it is possible to use a direction measure to test the number of times a prediction neural network predicts the direction of the predicted return movement correctly. Merton [1981] proposed a modified direction given by

$$\text{Modified Direction} = \frac{\sharp \text{ of correct up predictions}}{\sharp \text{ of times index up}} + \frac{\sharp \text{ of correct down predictions}}{\sharp \text{ of times index down}} - 1$$

In general, direction is defined as the number of times the prediction followed the up and down movement of the underlying stock/index. Harvey et al. [2002] and Castiglione et al. [2001] proposed the following direction measure

$$\xi = \frac{1}{|T|}\sum_{t\in T}\mathcal{H}(R_t\hat{R}_t) + 1 - (|R_t| + |\hat{R}_t|)$$

where $R_t = \frac{P_t}{P_{t-1}}$ is the percentage return on the underlying at time step $t \in T$, $\hat{R}_t$ is the forecast percentage return, T is the number of days in the validation period, and $\mathcal{H}(\cdot)$ is the Heaviside function. Alternatively, we can write the direction measure as

$$\xi = \frac{1}{|T|}\sum_{t\in T}\mathcal{H}(\Delta P_t\Delta\hat{P}_t) + 1 - (|\Delta P_t| + |\Delta\hat{P}_{i+1}|)$$

where $\Delta P_t = P_t - P_{t-1}$ is the price change at step $t \in T$ and P_{t-1} is assumed to be known. These two direction measures provide a summary of how well the predicted time series and the actual ones move together at any point in time. However, we want to implement measurement errors taking into account the simultaneous behaviour of trend and magnitude within the trend. Hence, to reflect this point when evaluating the performance of a forecasting model, Yao et al. [1996] used the correctness of trend and a paper profit. They proposed a profit based adjusted weight factor for backpropagation network training by adding a factor containing profit, direction, and time information to the error function.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

5.1.5.3 Time dependent direction profit

In order to take profit gain into account, Caldwell [1995] proposed a weighted directional symmetry (WDS) function which penalise more heavily the incorrectly predicted directions than the correct ones. The cumulative error function is given by

$$E_{WDS} = \frac{100}{P} \sum_{p=1}^{P} \phi(p)|O_p - d_p|$$

with

$$\phi(p) = \begin{cases} g \text{ if } \Delta d_p \times \Delta O_p \leqslant 0 \\ h \text{ otherwise} \end{cases}$$

where $\Delta d_p = d_p - d_{p-1}$ and $\Delta O_p = O_p - O_{p-1}$, and g and h are constants or some function of d_p. The values $g = 1.5$ and $h = 0.5$ were suggested. Yao et al. [1996] argued that the profit driven procedure was not only a function of the direction, but also of the amount of change. That is, the penalty on WDS weights should be increased in case of a wrongly forecasted direction for a big change of values, and the penalty should be further reduced if the direction is correctly forecasted for a big change in value. They proposed a modified directional profit (DP) adjustment factor defined as

$$\phi_{DP}(p) = \begin{cases} a_1 \text{ if } \Delta d_p \times \Delta O_p > 0 \text{ and } |\Delta d_p| \leqslant \sigma \\ a_2 \text{ if } \Delta d_p \times \Delta O_p > 0 \text{ and } |\Delta d_p| > \sigma \\ a_3 \text{ if } \Delta d_p \times \Delta O_p < 0 \text{ and } |\Delta d_p| \leqslant \sigma \\ a_4 \text{ if } \Delta d_p \times \Delta O_p < 0 \text{ and } |\Delta d_p| > \sigma \end{cases}$$

where σ is a threshold for the changes in sample values generally taken as the standard deviation of the training set

$$\sigma^2 = \frac{1}{P} \sum_{p=1}^{P} (d_p - \mu_d)^2$$

where μ_d is the mean of the target series. The values $a_1 = 0.5$, $a_2 = 0.8$, $a_3 = 1.2$, and $a_4 = 1.5$ were suggested. While the DLS model focus on a time factor $\phi_{DLS}(p)$ and the DP model focus on a profit factor $\phi_{DP}(p)$, Yao et al. [1998] [2000] combined the two approaches obtaining a time dependent directional profit (TDP) model given by

$$\phi_{TDP}(p) = \phi_{DLS}(p) \times \phi_{DP}(p)$$

The three models, DLS, DP, and TDP were tested and compared to the benchmarked OLS model on four major Asian stock indices and the US Dow Jones Industrials Index (DJ). They considered a one hidden layer network with learning rate $\eta = 0.25$ and momentum rate $\gamma = 0.9$. The structure of the neural network was $30 - 10 - 1$, that is, 30 input, 10 nodes in the hidden layer and one output. Thus, 30 consecutive days of data are fed to the network to forecast the index of the following day. Considering a paper profit measure for the model performance, they showed that the preference of DLS over OLS was of 60%, and that the preference of DP over OLS was of 80%. In this combined model, they obtained up to 27.6% excess annual return above the OLS model with at least 2.8% excess annual return in the worst case. Lu Dang Khoa et al. [2006] argued that big changes in prices between two consecutive periods are rare to occur, leading to change between two consecutive prices smaller than the standard deviation of the price. They replaced the inequality $|\Delta d_p| > \sigma$ with $\frac{\Delta d_p}{d_{p-1}} > a_5$ and obtained the following adjustment factor

$$\phi_{DPN}(p) = \begin{cases} a_1 \text{ if } \Delta d_p \times \Delta O_p > 0 \text{ and } \frac{\Delta d_p}{d_{p-1}} < a_5 \\ a_2 \text{ if } \Delta d_p \times \Delta O_p > 0 \text{ and } \frac{\Delta d_p}{d_{p-1}} > a_5 \\ a_3 \text{ if } \Delta d_p \times \Delta O_p < 0 \text{ and } \frac{\Delta d_p}{d_{p-1}} < a_5 \\ a_4 \text{ if } \Delta d_p \times \Delta O_p < 0 \text{ and } \frac{\Delta d_p}{d_{p-1}} > a_5 \end{cases}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

with $a_5 = 0.05$. They used that model to predict the $S\&P$ 500 stock index one month in the future without much improvements compared to the traditional FFN algorithm. They considered a mixed of input based on fundamental factors such as the prices of oil, the interest rates, inflation, and technical indicators such as volatility, relative strength index, directional index etc.

5.2 Introducing multilayer networks

While a single perceptron (see Section (5.1.1)) can only express linear decision surfaces, multilayer networks introduced in Section (5.1.3) are capable of expressing a rich variety of nonlinear decision surfaces. Since multiple layers of cascaded linear units can only produce linear functions, we need to introduce another unit to represent highly nonlinear functions. That is, we need a unit whose output is a nonlinear function of its inputs, but which is also a differentiable function of its input. We are first going to discuss in detail such a unit, and then describe how to apply the gradient descent (see Section (5.1.4)) in the case of multilayer networks.

5.2.1 Describing the problem

We consider a feedforward network (see Section (5.1.3)) with N_i input units, N_o output units and N_k hidden layers which can exhibit any desired feedforward connection pattern. We are also given a training set $\{(\overline{x}_1, \overline{d}_1), ..., (\overline{x}_P, \overline{d}_P)\}$ consisting of P ordered pairs of n- and m-dimensional vectors called the input and output patterns, respectively. Note, the training set may consists of sequential data, or time-varying input, which we then represent as $\{(\overline{x}_1, \overline{d}_1), ..., (\overline{x}_T, \overline{d}_T)\}$ with T ordered pairs. We assume that the primitive functions at each node of the network is continuous and differentiable, and that the weights of the edges are real numbers selected randomly so that the output $\overline{O}_p$ of the network is initially different from the target $\overline{d}_p$. The idea being to minimise the distance between $\overline{O}_p$ and $\overline{d}_p$ for $p = 1, ..., P$ by using a learning algorithm searching in a large hypothesis space defined by all possible weight values for all the units in the network.
There are different measures of error, and for simplicity we let the error function of the network be given by

$$E = \frac{1}{2} \sum_{p=1}^{P} ||\overline{O}_p - \overline{d}_p||_2^2$$

which is a sum of L_2 norms. After minimising the function with a training set, we can consider new unknown patterns and use the network to interpolate it. We use the backpropagation algorithm to find a local minimum to the error function by computing recursively the gradient of the error function and correcting the initial weights. Every one of the j output units of the network is connected to a node evaluating the function $\frac{1}{2}(O_{p,j} - d_{p,j})^2$ where $O_{p,j}$ and $d_{p,j}$ denote the jth component of the output vector $\overline{O}_p$ and the target vector $\overline{d}_p$, respectively. The outputs of the additional m nodes are collected at a node which adds them up and gives the sum

$$E_p = \sum_{j=1}^{N_o} \frac{1}{2}(O_{p,j} - d_{p,j})^2$$

as its output. The same network extension is built for each pattern $\overline{d}_p$. We can then collect all the quadratic errors and output their sum, obtaining the total error for a given training set

$$E = \sum_{p=1}^{P} E_p$$

Since E is computed exclusively through composition of the node functions, it is a continuous and differentiable function of the q weights $w_1, .., w_q$ in the network. Therefore, we can minimise the total error E by using an iterative process of gradient descent (see Section (5.1.4)), where we need to calculate the gradient

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\nabla E = (\frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_q})$$

and each weight is updated with the increment

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \text{ for } i = 1, .., q$$

where η is a learning constant, that is, a proportionality parameter defining the step length of each iteration in the negative gradient direction. Hence, once we have a method for computing the gradient, we can adjust the weights iteratively until we find a minimum of the error function where $\nabla E \approx 0$.

***Remark* 5.2.1** *In the case of multilayer networks, the error surface can have multiple local minima. Hence, the minimisation by steepest descent will only produce a local minimum of the error function, and not necessarily the global minimum error.*

Note, the operations involved in the computation of the gradients are usually represented as a computation graph since, as discussed (5.1.3.2), a neural network is essentially a mathematical expression involving composite functions, and can be represented as a graph (see Bengio [2012]). The graph abstraction (see Section (13.1)) allows one to easily construct arbitrary networks, evaluate their predictions and compute gradients for their parameters through the chain rule of differentiation.

5.2.2 Describing the algorithm

For simplicity of exposition we first describe the algorithm in the case where there is no hidden layer and the training set consists of a single input-output pair ($P = 1$). We assume a set of artificial neurons $\{f_{i,j}\}$, each receiving $\{x_i\}_{i=1}^{N_i}$ input and computing $\{O_j\}_{j=1}^{N_o}$ output, and we focus our attention on one of the weights, $w_{i,j}$, going from input i to neuron j in the network. We let W denote the $N_i \times N_o$ weight matrix with element $w_{i,j}$ at the ith row and the jth column, and we let $w(j)$ be the $N_i \times 1$ row vector for the jth column. In the back-propagation algorithm, given the input vector $x = (x_1, .., x_{N_i})$, the activation function (also called net_j) for the jth neuron satisfies the weighted sum

$$A_j(x, w) = x \cdot w(j) = \sum_{i=1}^{N_i} x_i w_{i,j} \text{ , } j = 1, .., N_o$$

which only depends on the inputs and the weights $w_{i,j}$ from input i to neuron j. We let the output function (or threshold box, or activation function) be a function g of the activation function, getting

$$O_j(x, w) = g(\tilde{A}_j(x, w)) \text{ , } j = 1, .., N_o$$

where $\tilde{A}_j(x, w) = A_j(x, w) + b_j$ and b_j is a bias value. In compact form, the output vector of all units is

$$O(x, w) = g(xW)$$

using the convention that we apply the function $g(\cdot)$ to each component of the argument vector. The simplest output function is the identity function. When using a threshold activation function, if the previous output of the neuron is greater than the threshold of the neuron, the output of the neuron will be one, and zero otherwise. Further, to simplify computation, the threshold can be equated to an extra weight. The error being the difference between the actual and the desired output, it only depends on the weights. Hence, to minimise the error by adjusting the weights, we define the error function for the output of each neuron. The error function for the jth neuron satisfies

$$E_j(x, w, d) = \big(O_j(x, w) - d_j\big)^2 \text{ , } j = 1, .., N_o$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where d_j is the jth element the desired target vector $\overline{d}$. In that setting, the total error of the network is simply the sum of the errors of all the neurons in the output layer

$$E(x,w,d) = \frac{1}{2}||\overline{O}(x,w) - \overline{d}(x)||_2^2 = \frac{1}{2}\sum_{j=1}^{N_o} E_j(x,w,d)$$

To minimise the error, we will update the weights of the network so that the expected error in the next iteration is lower using the method of gradient descent (see Section (5.1.4)). Each weight is updated by using the increment

$$\Delta w_{i,j} = w_{i,j}^{l+1} - w_{i,j}^{l} = -\eta \frac{\partial E}{\partial w_{i,j}^{l}}$$

where l is the iteration counter, and $\eta \in (0,1]$ is a learning rate. Using vector notation, for the jth neuron, we get

$$\Delta w(j) = -\eta\big(O_j(x,w) - d_j\big)\nabla_{w(j)} O_j(x,w)$$

The size of the adjustment depends on η and on the contribution of the weight to the error of the function. The adjustment will be largest for the weight contributing the most to the error. We repeat the process until the error is minimal.

5.2.3 Describing the nonlinear transformation

We use the chain rule to compute the gradient of the error function. Since this method requires computation of the gradient of the error function at each iteration step, we must guarantee the continuity and differentiability of the error function. One way forward is to use the sigmoid, or logistic function, a real function $f_c : \mathbb{R} \to (0,1)$ defined as

$$f_c(x) = \frac{1}{1+e^{-cx}}$$

where the constant c can be chosen arbitrarily, and its reciprocal $\frac{1}{c}$ is called the temperature parameter. For $c \to \infty$ the sigmoid converges to a step function at the origin. Further, $\lim_{x\to\infty} f_c(x) = 1$, $\lim_{x\to-\infty} f_c(x) = 0$, and $\lim_{x\to 0} f_c(x) = \frac{1}{2}$. The first derivative of the sigmoid with respect to x is

$$\frac{d}{dx} f_c(x) = \frac{ce^{-cx}}{(1+e^{-cx})^2} = cf_c(x)(1-f_c(x)) \tag{5.2.10}$$

and the second derivative of the sigmoid with respect to x is

$$\frac{d^2}{dx^2} f_c(x) = cf_c^{'}(x) - 2cf_c(x)f_c^{'}(x)$$

where $f_c^{'}(x) = \frac{d}{dx} f_c(x)$. To get a symmetric output function, we can consider the symmetrical sigmoid $f_s(x)$ defined as

$$f_s(x) = 2f_1(x) - 1 = \frac{1-e^{-x}}{1+e^{-x}}$$

which is the *hyperbolic tangent* for the argument $\frac{x}{2}$, written $tanh(\frac{x}{2})$. An alternative solution is simply to linearly translate the domain of definition of the logistic function in the range $[b_L, b_H]$ where b_L is the lower bound and b_H the upper bound. Hence, the real function $f_{b,c} : \mathbb{R} \to (b_L, b_H)$ is defined as

$$f_{b,c}(x) = b_L + (b_H - b_L)f_c(x)$$

with $\lim_{x\to\infty} f_c(x) = b_H$, $\lim_{x\to-\infty} f_c(x) = b_L$, and $\lim_{x\to 0} f_c(x) = \frac{1}{2}(b_L + b_H)$. The derivative of this function with respect to x is

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\frac{d}{dx}f_{b,c}(x) = (b_H - b_L)\frac{d}{dx}f_c(x) = (b_H - b_L)cf_c(x)(1 - f_c(x))$$

Hence, in order to get a symmetric output function we can set $b_L = -1.5$ and $b_H = 1.5$. Example of translated logistic function and its derivative are given in Figure (5.6) for $b_L = -0.6$ and $b_H = 0.6$ with $c = \frac{1}{p}$ for $p = 0.5, 1, 1.5$. Several other output functions can be used and have been proposed in the back-propagation algorithm. However, smoothed output function can lead to local minima in the error function which would not be there if the Heavised function had been used. Further, the slope of the sigmoid function given in Equation (5.2.10) takes values in the range $[0, \frac{1}{4}c]$ where the maximum is reached at $x = 0$. Thus, the steepest slope of the sigmoid function occurs at the origin, and depending on the size of the constant c, the slope can become greater than 1, leading to gradient explosion. In addition, the derivative of the sigmoid function is asymptotically zero for large values of x, such that the gradient will decay much more when the net value is taking large of small values. The decaying or explosion of the gradient are both called the vanishing gradient problem.

Figure 5.6: Translated logistic function and its derivative for $b_L = -0.6$ and $b_H = 0.6$.

5.2.4 A simple example

To explain the method, we first consider a two-layers ANN model, where the most common output function used with a threshold is the sigmoidal function

$$O_j(x, w) = \frac{1}{1 + e^{-A_j(x,w)}}$$

with the limits $\lim_{A_j \to \infty} = 1$, $\lim_{A_j \to -\infty} = 0$, and $\lim_{A_j \to 0} = \frac{1}{2}$ allowing for a smooth transition between the low and high output of the neuron. Note, A_j is sometimes denoted net_j since it is the net input coming into output unit j from a unit i that projects to it. We compute the gradient of the total error with respect to the weight $w_{i,j}$, noted $\nabla E(w_{i,j})$, where it is understood that the subscripts i and j are fixed integers within $i = 1, ..., N_i$ and $j = 1, .., N_o$. From the linearity of the total error function, the gradient is given by

$$\begin{aligned}\frac{\partial E}{\partial w_{i,j}} &= \frac{1}{2}\sum_{j=1}^{N_o}\frac{\partial E_j}{\partial w_{i,j}}, i = 1, ..., N_i \\ &= \frac{1}{2}\sum_{j=1}^{N_o}\frac{\partial E_j}{\partial O_j}\frac{\partial O_j}{\partial w_{i,j}}\end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where the subscripts i and j are fixed integers in the partial derivatives. We first differentiate the total error with respect to the output function

$$\frac{\partial E_j}{\partial O_j} = (O_j - d_j)$$

and then differentiate the output function with respect to the weights (see Equation (5.2.10))

$$\frac{\partial O_j}{\partial w_{i,j}} = \frac{\partial O_j}{\partial A_j}\frac{\partial A_j}{\partial w_{i,j}} = O_j(1 - O_j)x_i$$

since $\frac{\partial A_j}{\partial w_{i,j}} = x_i$. Putting terms together, the adjustment to each weight becomes

$$\Delta w_{i,j} = -\eta(O_j - d_j)O_j(1 - O_j)x_i = \eta(d_j - O_j)O_j(1 - O_j)x_i$$

We can use the above equation to train an ANN with two layers. Given a training set with p input-output pairs, the error function can be computed by creating p similar networks and adding the outputs of all of them to obtain the total error of the set.

5.3 Multi-layer back propagation

In the case of a multilayer network, again we consider a single input-output pair and assume a set of artificial neurons $\{f_{i,j,k}\}_{k=0}^{K}$ where the multilayer subscript $k = 0$ corresponds to the set of inputs $\{x_i\}_{i=1}^{N_i}$, and the remaining subscript k corresponds to the set of outputs $\{y_i\}_{i=1}^{N_{o,k}}$, where $N_{o,k}$ is the number of output in the k-th layer.
In that setting, we define the output function of the jth node for the k-th layer as

$$O_{j,k}(x, w) = g(\tilde{A}_{j,k-1}(x, w)) \text{ , } j = 1, .., N_{o,k}$$

where $\tilde{A}_{j,k-1}(x, w) = A_{j,k-1}(x, w) + b_{j,k}$ with the bias $b_{j,k}$. In order to simplify notation we will sometimes refer to $net_{j,k} = \tilde{A}_{j,k-1}(x, w)$ as the net value. We let the corresponding activation function satisfies

$$A_{j,k-1}(x, w) = \sum_{i=1}^{N_{o,k-1}} O_{i,k-1}(x, w)w_{i,j}^{k-1} \text{ and } O_{j,0}(x, w) = A_{j,-1}(x, w) = \sum_{i=1}^{N_i} x_i$$

where $w_{i,j}^{k-1}$ is the weight going from the ith node in layer $k - 1$ to the jth node in layer k. Thus,

$$\frac{\partial net_{j,k}}{\partial w_{i,j}^{k-1}} = \frac{\partial A_{j,k-1}(x, w)}{\partial w_{i,j}^{k-1}} = O_{i,k-1}(x, w) \tag{5.3.11}$$

Note, since $\frac{d\tilde{A}_{j,k-1}(x,w)}{dA_{j,k-1}(x,w)} = 1$, we get $\frac{\partial O_{j,k}(x,w)}{\partial A_{j,k-1}(x,w)} = \frac{\partial O_{j,k}(x,w)}{\partial \tilde{A}_{j,k-1}(x,w)}$ where

$$\frac{\partial O_{j,k}(x, w)}{\partial net_{j,k}} = \frac{\partial O_{j,k}(x, w)}{\partial \tilde{A}_{j,k-1}(x, w)} = \frac{\partial}{\partial A_{j,K-1}} g(\tilde{A}_{j,K-1}(x, w))$$

The error function for the sth neuron on the Kth-layer satisfies

$$E_s(x, w, d) = \left(O_{s,K}(x, w) - d_s\right)^2 \text{ , } s = 1, ..., N_{o,K}$$

where d_s is the sth element of the target vector d. The total error of the network is the sum of the errors of all the neurons in the output layer

$$E(x, w, d) = \frac{1}{2}\|O(x, w) - d\|^2 = \frac{1}{2}\sum_{s=1}^{N_{o,K}} E_s(x, w, d)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The gradient of the total error with respect to the weight $w_{i,j}^{k-1}$ is given by

$$\begin{aligned}\nabla E(w_{i,j}^{k-1}) = \frac{\partial}{\partial w_{i,j}^{k-1}} E(x,w,d) &= \frac{\partial E(x,w,d)}{\partial net_{j,k}} \frac{\partial net_{j,k}}{\partial w_{i,j}^{k-1}} \\ &= \frac{\partial E(x,w,d)}{\partial O_{j,k}(x,w)} \frac{\partial O_{j,k}(x,w)}{\partial net_{j,k}} \frac{\partial net_{j,k}}{\partial w_{i,j}^{k-1}}\end{aligned} \tag{5.3.12}$$

We will call $e_{j,k} = \frac{\partial E(x,w,d)}{\partial O_{j,k}(x,w)}$ the pre-error signal, for the jth unit on the kth-layer, and

$$\delta_{j,k} = \frac{\partial E(x,w,d)}{\partial net_{j,k}} = e_{j,k} \frac{\partial O_{j,k}(x,w)}{\partial net_{j,k}}$$

the error signal, and we will rewrite the gradient of the total error as

$$\begin{aligned}\nabla E(w_{i,j}^{k-1}) &= \delta_{j,k} O_{i,k-1}(x,w) \\ &= e_{j,k} \frac{\partial O_{j,k}(x,w)}{\partial net_{j,k}} O_{i,k-1}(x,w)\end{aligned} \tag{5.3.13}$$

with stochastic gradient descent rule given by

$$\Delta w_{i,j}^{k-1} = -\eta \nabla E(w_{i,j}^{k-1}) \tag{5.3.14}$$

5.3.1 The output layer

We start with the output layer K and compute the gradient $\nabla E(w_{i,j}^{k-1})$ for the weight $w_{i,j}^{k-1}$ going from the ith node in layer $(K-1)$ to the jth node in layer K. From the definition of the total error, the gradient satisfies

$$\nabla E(w_{i,j}^{k-1}) = \frac{\partial}{\partial w_{i,j}^{k-1}} E(x,w,d) = (O_{j,K} - d_j) \frac{\partial}{\partial w_{i,j}^{k-1}} (O_{j,K} - d_j)$$

since the subsripts i and j are fixed integers. Expending the output function $O_{j,K}$, we get

$$\frac{\partial}{\partial w_{i,j}^{k-1}} E(x,w,d) = (O_{j,K} - d_j) \frac{\partial}{\partial w_{i,j}^{k-1}} g\big(A_{j,K-1}(x,w) + b_{j,K}\big)$$

Using once again the chain rule, we obtain

$$\frac{\partial}{\partial w_{i,j}^{k-1}} E(x,w,d) = (O_{j,K} - d_j) \frac{\partial}{\partial A_{j,K-1}} g(\tilde{A}_{j,K-1}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-1}} A_{j,K-1}(x,w)$$

where $\tilde{A}_{j,K-1}(x,w) = A_{j,K-1}(x,w) + b_{j,K}$. The only term that depends on $w_{i,j}^{k-1}$ in the activation function $A_{j,K-1}(x,w)$ is $O_{i,K-1}(x,w) w_{i,j}^{k-1}$, and the rest of the sum will zero out after derivation. Therefore, the gradient becomes

$$\begin{aligned}\frac{\partial}{\partial w_{i,j}^{k-1}} E(x,w,d) &= (O_{j,K} - d_j) \frac{\partial}{\partial A_{j,K-1}} g(\tilde{A}_{j,K-1}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-1}} O_{i,K-1}(x,w) w_{i,j}^{k-1} \\ &= (O_{j,K} - d_j) \frac{\partial}{\partial A_{j,K-1}} g(\tilde{A}_{j,K-1}(x,w)) O_{i,K-1}(x,w)\end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

and we obtain the adjustment to each weight $\Delta w_{i,j}^{k-1}$.
We define the pre-error signal as

$$e_{j,K} = \frac{\partial}{\partial O_{j,K}} E = (O_{j,K} - d_j)$$

and rewrite the partial derivative as

$$\frac{\partial}{\partial w_{i,j}^{k-1}} E(x,w,d) = e_{j,K} \frac{\partial}{\partial A_{j,K-1}} g(\tilde{A}_{j,K-1}(x,w)) O_{i,K-1}(x,w)$$

Further, setting the error signal as

$$\delta_{j,K} = \frac{\partial}{\partial net_{j,K}} E$$

where $net_{j,K} = \tilde{A}_{j,K-1}(x,w)$, we get

$$\delta_{j,K} = e_{j,K} \frac{\partial O_{j,K}(x,w)}{\partial net_{j,K}} = e_{j,K} \frac{\partial}{\partial A_{j,K-1}} g(\tilde{A}_{j,K-1}(x,w)) \tag{5.3.15}$$

and the gradient simplifies to

$$\nabla E(w_{i,j}^{k-1}) = \delta_{j,K} O_{i,K-1}(x,w)$$

so that the stochastic gradient descent rule for output units becomes

$$\Delta w_{i,j}^{k-1} = -\eta \frac{\partial}{\partial w_{i,j}^{k-1}} E = -\eta \delta_{j,K} O_{i,K-1}(x,w)$$

We observe that the weight update rule $\Delta w_{i,j}^{k-1}$ is a multiplication of the error introduced to the output times the gradient of the output function of the current neurons input times this neurons input.

5.3.2 The first hidden layer

The next step is to consider the hidden layer $(K-1)$ and compute the gradient for the weight $w_{i,j}^{k-2}$ going from the ith node on layer $(K-2)$ to the jth node in layer $(K-1)$. Note, since the subscripts i and j are taken, for notation purpose, we use the subscript s to represent the nodes on the K-th layer. The gradient $\nabla E(w_{i,j}^{k-2})$ for the weight $w_{i,j}^{k-2}$ becomes

$$\nabla E(w_{i,j}^{k-2}) = \frac{\partial}{\partial w_{i,j}^{k-2}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial w_{i,j}^{k-2}} (O_{s,K} - d_s)$$

which gives

$$\frac{\partial}{\partial w_{i,j}^{k-2}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial w_{i,j}^{k-2}} g\big(A_{s,K-1}(x,w) + b_{s,K}\big)$$

Using once again the chain rule, we obtain

$$\frac{\partial}{\partial w_{i,j}^{k-2}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-2}} A_{s,K-1}(x,w)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $A_{s,K-1}(x,w) = \sum_{j=1}^{N_{o,K-1}} O_{j,K-1}(x,w)w_{j,s}^{k-1}$. The only term that depends on $w_{i,j}^{k-2}$ in the activation function $A_{s,K-1}(x,w)$ is $O_{j,K-1}(x,w)w_{j,s}^{k-1}$, and the rest of the sum will zero out after derivation. Therefore, the gradient simplifies to

$$\frac{\partial}{\partial w_{i,j}^{k-2}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,k-1}} g(\tilde{A}_{s,K-1}(x,w)) w_{j,s}^{k-1} \frac{\partial}{\partial w_{i,j}^{k-2}} O_{j,K-1}(x,w)$$

which becomes

$$\frac{\partial}{\partial w_{i,j}^{k-2}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) w_{j,s}^{k-1} \frac{\partial}{\partial w_{i,j}^{k-2}} g(A_{j,K-2}(x,w) + b_{j,K-1})$$

Using once again the chain rule, we obtain

$$\frac{\partial}{\partial w_{i,j}^{k-2}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) w_{j,s}^{k-1} \frac{\partial}{\partial A_{j,K-2}} g(\tilde{A}_{j,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-2}} A_{j,K-2}(x,w)$$

where $A_{j,K-2}(x,w) = \sum_{i=1}^{N_{o,K-2}} O_{i,K-2}(x,w)w_{i,j}^{k-2}$. The only term that depends on $w_{i,j}^{k-2}$ in the activation function $A_{j,K-2}(x,w)$ is $O_{i,K-2}(x,w)w_{i,j}^{k-2}$, and the rest of the sum will zero out after derivation. Therefore, given the pre-error signal $e_{s,K} = (O_{s,K} - d_s)$, we get

$$\begin{aligned}
& \frac{\partial}{\partial w_{i,j}^{k-2}} E(x,w,d) \\
= & \sum_{s=1}^{N_{o,K}} e_{s,K} \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) w_{j,s}^{k-1} \frac{\partial}{\partial A_{j,K-2}} g(\tilde{A}_{j,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-2}} O_{i,K-2}(x,w) w_{i,j}^{k-2} \\
= & \sum_{s=1}^{N_{o,K}} e_{s,K} \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) w_{j,s}^{k-1} \frac{\partial}{\partial A_{j,K-2}} g(\tilde{A}_{j,K-2}(x,w)) O_{i,K-2}(x,w)
\end{aligned}$$

and we obtain the adjustment to each weight $\Delta w_{i,j}^{k-2}$. Thinking in terms of the pre-error signal, the error from the previous layer $e_{j,K-1} = \frac{\partial}{\partial O_{j,K-1}} E$ is given by

$$\begin{aligned}
e_{j,K-1} &= \sum_{s=1}^{N_{o,K}} e_{s,K} \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) w_{j,s}^{k-1} \\
&= \sum_{s=1}^{N_{o,K}} e_{s,K} \frac{\partial}{\partial net_{s,K}} O_{s,K}(x,w) w_{j,s}^{k-1} \\
&= \sum_{s=1}^{N_{o,K}} \delta_{s,K} w_{j,s}^{k-1} \text{ from Equation (5.3.15)}
\end{aligned}$$

***Remark* 5.3.1** *Some authors refer to the term Downstream of unit j to describe all units whose direct inputs include the output of unit j.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Hence, we see that the error from the previous layer is scaled down in proportion to the amount of how the previous layer influenced it. Again, the weight update rule $\Delta w_{i,j}^{k-2}$ is a multiplication of the error introduced to the output times the gradient of the output function of the current neurons input times this neurons input. That is,

$$\frac{\partial}{\partial w_{i,j}^{k-2}} E(x,w,d) = e_{j,K-1} \frac{\partial}{\partial A_{j,K-2}} g(\tilde{A}_{j,K-2}(x,w)) O_{i,K-2}(x,w)$$

Setting the error signal $\delta_{j,K-1} = \frac{\partial}{\partial net_{j,K-1}} E$, where $net_{j,K-1} = \tilde{A}_{j,K-2}(x,w)$, as

$$\delta_{j,K-1} = e_{j,K-1} \frac{\partial O_{j,K-1}(x,w)}{\partial net_{j,K-1}} = e_{j,K-1} \frac{\partial}{\partial A_{j,K-2}} g(\tilde{A}_{j,K-2}(x,w)) \tag{5.3.16}$$

the gradient simplifies to

$$\nabla E(w_{i,j}^{k-2}) = \delta_{j,K-1} O_{i,K-2}(x,w)$$

so that the stochastic gradient descent rule for output units becomes

$$\Delta w_{i,j}^{k-2} = -\eta \frac{\partial}{\partial w_{i,j}^{k-2}} E = -\eta \delta_{j,K-1} O_{i,K-2}(x,w)$$

By iteration, we repeat this procedure until we reach $\Delta w_{i,j}^{1}$. Note, we differentiate the error function with respect to the bias value $b_{j,K}$ in the same way.

5.3.3 The next hidden layer

Going one step backward, we consider the hidden layer $(K-2)$ and compute the gradient for the weight $w_{i,j}^{k-3}$ going from the ith node on layer $(K-3)$ to the jth node in layer $(K-2)$. We will also assume that it corresponds to the input layer. Note, since the subscripts i and j are taken, for notation purpose, we use the subscript s to represent the nodes on the K-th layer and t to represent the nodes on the $(K-1)$-th layer. We get

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial w_{i,j}^{k-3}} (O_{s,K} - d_s)$$

which gives

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial w_{i,j}^{k-3}} g\big(A_{s,K-1}(x,w) + b_{s,K}\big)$$

Using once again the chain rule, we obtain

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} A_{s,K-1}(x,w)$$

where $A_{s,K-1}(x,w) = \sum_{t=1}^{N_{o,K-1}} O_{t,K-1}(x,w) w_{t,s}^{k-1}$. Replacing in the equation we get

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \sum_{t=1}^{N_{o,K-1}} \frac{\partial}{\partial w_{i,j}^{k-3}} O_{t,K-1}(x,w) w_{t,s}^{k-1}$$

which gives

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \sum_{t=1}^{N_{o,K-1}} \frac{\partial}{\partial w_{i,j}^{k-3}} g\big(A_{t,K-2}(x,w) + b_{t,K-1}\big) w_{t,s}^{k-1}$$

Using the chain rule, we obtain

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) =$$
$$\sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \sum_{t=1}^{N_{o,K-1}} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} A_{t,K-2}(x,w) w_{t,s}^{k-1}$$

where $A_{t,K-2}(x,w) = \sum_{j=1}^{N_{o,K-2}} O_{j,K-2}(x,w) w_{j,t}^{k-2}$. The only term that depends on $w_{i,j}^{k-3}$ in the activation function $A_{t,K-2}(x,w)$ is $O_{j,K-2}(x,w) w_{j,t}^{k-2}$, and the rest of the sum will zero out after derivation. Therefore, given the pre-error signal $e_{s,K} = (O_{s,K} - d_s)$, we get

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} e_{s,K} \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \sum_{t=1}^{N_{o,K-1}} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} O_{j,K-2}(x,w) w_{j,t}^{k-2} w_{t,s}^{k-1}$$

which becomes

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) =$$
$$\sum_{s=1}^{N_{o,K}} e_{s,K} \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \sum_{t=1}^{N_{o,K-1}} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} g(A_{j,K-3}(x,w) + b_{j,K-2}) w_{j,t}^{k-2} w_{t,s}^{k-1}$$

Setting

$$\hat{e}_{t,K-1} = e_{s,K} \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) w_{t,s}^{k-1}$$

we get

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} \sum_{t=1}^{N_{o,K-1}} \hat{e}_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} g(A_{j,K-3}(x,w) + b_{j,K-2}) w_{j,t}^{k-2}$$

Note, from linearity we can interchange the summation operators and rewrite the above equation as

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{t=1}^{N_{o,K-1}} \sum_{s=1}^{N_{o,K}} \hat{e}_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} g(A_{j,K-3}(x,w) + b_{j,K-2}) w_{j,t}^{k-2}$$

We then recover the pre-error term $e_{t,K-1} = \frac{\partial}{\partial O_{t,K-1}} E$ as

$$e_{t,K-1} = \sum_{s=1}^{N_{o,K}} \hat{e}_{s,K} = \sum_{s=1}^{N_{o,K}} e_{s,K} \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) w_{t,s}^{k-1}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

such that the equation simplifies to

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{t=1}^{N_{o,K-1}} e_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} g(A_{j,K-3}(x,w) + b_{j,K-2}) w_{j,t}^{k-2}$$

Using once again the chain rule, we obtain

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{t=1}^{N_{o,K-1}} e_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial A_{j,K-3}} g(\tilde{A}_{j,K-3}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} A_{j,K-3}(x,w) w_{j,t}^{k-2}$$

where $A_{j,K-3}(x,w) = \sum_{i=1}^{N_{o,K-3}} O_{i,K-3}(x,w) w_{i,j}^{k-3}$. The only term that depends on $w_{i,j}^{k-3}$ in the activation function $A_{j,K-3}(x,w)$ is $O_{i,K-3}(x,w) w_{i,j}^{k-3}$, and the rest of the sum will zero out after derivation. Therefore, the gradient simplifies to

$$\begin{aligned}
& \frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) \\
= & \sum_{t=1}^{N_{o,K-1}} e_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial A_{j,K-3}} g(\tilde{A}_{j,K-3}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} O_{i,K-3}(x,w) w_{i,j}^{k-3} w_{j,t}^{k-2} \\
= & \sum_{t=1}^{N_{o,K-1}} e_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial A_{j,K-3}} g(\tilde{A}_{j,K-3}(x,w)) O_{i,K-3}(x,w) w_{j,t}^{k-2}
\end{aligned}$$

Writing the error from the previous layer in terms of the Downstream of unit j as

$$\begin{aligned}
e_{j,K-2} &= \sum_{t=1}^{N_{o,K-1}} e_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) w_{j,t}^{k-2} \\
&= \sum_{t=1}^{N_{o,K-1}} \delta_{t,K-1} w_{j,t}^{k-2} \text{ From Equation (5.3.16)}
\end{aligned}$$

where $e_{j,K-2} = \frac{\partial}{\partial O_{j,K-2}} E$. Then, the gradient becomes

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = e_{j,K-2} \frac{\partial}{\partial A_{j,K-3}} g(\tilde{A}_{j,K-3}(x,w)) O_{i,K-3}(x,w)$$

which corresponds to the multiplication of the error introduced to the output times the gradient of the output function of the current neurons input times this neurons input. Further, setting the error signal $\delta_{j,K-2} = \frac{\partial}{\partial net_{j,K-2}} E$, where $net_{j,K-2} = \tilde{A}_{j,K-3}(x,w)$, as

$$\delta_{j,K-2} = e_{j,K-2} \frac{\partial O_{j,K-2}(x,w)}{\partial net_{j,K-2}} = e_{j,K-2} \frac{\partial}{\partial A_{j,K-3}} g(\tilde{A}_{j,K-3}(x,w)) \tag{5.3.17}$$

the gradient simplifies to

$$\nabla E(w_{i,j}^{k-3}) = \delta_{j,K-2} O_{i,K-3}(x,w)$$

and the stochastic gradient descent rule for the next hidden layer units becomes

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\Delta w_{i,j}^{k-3} = -\eta \frac{\partial}{\partial w_{i,j}^{k-3}} E = -\eta \delta_{j,K-2} O_{i,K-3}(x, w)$$

This is the general rule for updating internal unit weights in arbitrary multilayer networks. Hence, the error signal travels from the output layer to the input layer. Further, the weights influence the error by some degree, and they must be taken into consideration when propagating the error.

5.3.4 Some remarks

Since the error surface for multipayer networks may contain many different local minima, the backpropagation algorithm can only converge toward some local minima in E which is not necessarily the global minimum error. Nonetheless, when the gradient descent falls into a local minimum with respect to one of these weights, it will not necessarily be in a local minimum with respect to other weights. Hence, higher dimensions might provide escape routes to the steepest descent to continue searching the space of possible network weights. In addition, the sigmoid threshold function being approximately linear when the weights are close to zero, we can initialise the network weights to values near zero so that in the early steps the network will represent a very smooth function approximately linear in its inputs. There exists several heuristic to avoid being stuck in a local minima such as adding a momentum term to the weight-update rule or using a stochastic gradient descent. One of the best approach is to train multiple networks using the same data, but initialising each network with different random weights. One can then select the best network according to one of these two methods

1. select the network with the best performance over a separate validation data set.
2. all networks can be retained and treated as a committee of networks whose output is the weighted average of the individual network outputs.

Various authors investigated the backpropagation algorithm to find out which function classes could be described by which types of networks. Three general results are known

- Boolean functions: every boolean function can be represented exactly by some network with two layers of units, although the number of hidden units required grows exponentially in the worst case with the number of network inputs.
- Continuous functions: every bounded continuous function can be approximated with arbitrarily small error (under a finite norm) by a network with two layers of units (see Cybenko [1989]).
- Arbitrary functions: any function can br approximated to arbitrary accuracy by a network with three layers of units (see Cybenko [1988]). This is because any function can be approximated by a linear combination of many localised functions having value 0 everywhere except for some small region, and that two layers of sigmoid units are sufficient to produce good local approximations.

The hypothesis space for backpropagation is the n-dimensional Euclidean space of the n network weights. Further, as opposed to decision tree where the hypothesis space is discrete , it is continuous for backpropagation leading to a well-defined error gradient. Also, the inductive bias of backpropagation learning is characterised by smooth interpolation between data points. An important property of backpropagation is its ability to discover useful intermediate representations at the hidden unit layers inside the network. This ability of multilayer networks to automatically discover useful representations at the hidden layers is a key feature of ANN learning as it provides extra degree of flexibility to invent features not explicitly introduced by the human designer.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

5.4 Summarising the feedforward ANN

As observed in Section (5.3), when running backpropagation, the error signal travels from the output layer to the input layer. Given a hidden layer with subscript k, and focusing on neuron with index j, we define the pre-error signal $e_{j,k}$ and the error signal $\delta_{j,k}$ as

$$\begin{aligned} e_{j,k} &= \frac{\partial}{\partial O_{j,k}} E \\ \delta_{j,k} &= \frac{\partial}{\partial net_{j,k}} E = \frac{\partial}{\partial \tilde{A}_{j,k-1}(x,w)} E \\ &= e_{j,k} \frac{\partial}{\partial net_{j,k}} O_{k,k} \end{aligned}$$

and the stochastic gradient descent rule for the next hidden layer units becomes

$$\Delta w_{i,j} = -\eta \delta_{j,k} O_{i,k-1}(x,w)$$

where i is the subscript of a neuron in the hidden layer $(k-1)$.
Using these results, we are going to summarise the feedforward neural network by considering the multilayer perceptron (MLP). Note, we slightly modify our notation from Section (5.3).

5.4.1 Forward pass

We consider an MLP consisting of K hidden layers with N_i input units, $N_{o,k}$ output units in the k-th hidden layer and $N_{o,K}$ output units in the output layer. In each unit of the hidden layer, or in the output layer, we first calculate the weighted sum of the incoming values, which is referred to as the *net value* of the input unit. The *activation function*, denoted by f, is then applied to the net value. The value of the activation function, y, is the output of the unit. The activation function is required to be bounded, differentiable and monotonous. The two most common functions used in machine learning are the *hyperbolic tangent* and the *logistic sigmoid* functions, which are both nonlinear (see Section (5.2.3)). This important feature makes it possible for the network to model nonlinear equations. As the number of hidden layers increases, the network can approximate more complex nonlinear functions. Methods using a network with a large number of hidden layers are referred to as *deep learning* network. Since the input domain of the activation function is infinite, while the output domain is finite, the activation functions are also termed as *squashing functions*.
Letting h be the index of the first hidden unit, the net value and output function satisfy

$$\begin{aligned} net_{h,1} &= \sum_{i=1}^{N_i} w_{ih} x_i \\ y_h &= f(net_{h,1}) \end{aligned} \tag{5.4.18}$$

Considering two adjacent layers, denoted by $k-1$ and k, with index h and h', respectively, the summation and the activation process are similar to that of Equation (5.4.18). They are given by

$$\begin{aligned} net_{h',k} &= \sum_{h=1}^{N_{o,k-1}} w_{hh'} y_h \\ y_{h'} &= f(net_{h',k}) \end{aligned} \tag{5.4.19}$$

The net value and activation function of the output layer are calculated in the same way as those in the hidden layer. We let s be the index ranging over the output layer, and $N_{o,K-1}$ is the number of neurons in the $(K-1)$-th hidden layer closest to the output layer. We get

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$
\begin{aligned}
net_{s,K} &= \sum_{h=1}^{N_{o,K-1}} w_{hs} y_h \\
y_s &= f(net_{s,K})
\end{aligned}
\tag{5.4.20}
$$

5.4.2 Backward pass

In general, after a forward pass the network output is not the target input, and we need to measure the distance between the actual output and the target output to serve as a measurement of the network performance. This distance is called the *error function*. Given the input-target pair (x, z), and the network output y, then the error function is given by

$$
E(x, z) = \frac{1}{2} \sum_{s:output} (y_s - z_s)^2, \tag{5.4.21}
$$

where $y = (y_1, \cdots, y_s, \cdots, y_K)$, $z = (z_1, \cdots, z_s, \cdots, z_K)$, and $N_{o,K}$ is the number of units in the output layer. As long as the error function is larger than a given tolerence level, we need to modify the weights of the network to further decrease the measure E. The algorithm designed for stepwise updating the weights that minimise the error function is called the *training algorithm*, or *learning algorithm* of the network. It is natural to relate the weights updates to the *gradient*, which is a vector of derivatives of the error function with respect to all the weights. It costs the network certain number of steps to reach a tolerably small error. Putting all the weights in a vector, the weight vector at step n is denoted by w^n, which is then updated by an amount Δw^n. We denote the gradient as $\nabla E = \frac{\partial E}{\partial w^n}$ and apply the *gradient descent* repeatedly using the chain rule (see Section (5.1.4)). By working backward, from computing the derivative with respect to the output layer to computing the derivatives with respect to all the internal weights, the error is back propagated in the network. This procedure is called the standard *back propagation*, discovered independently by different researchers (see Werbos [1974], Parker [1985], Rumelhart et al. [1986b]). In standard back propagation, the weight update of the n-th step is

$$
\Delta w^n = -\eta \frac{\partial E}{\partial w^n}, \tag{5.4.22}
$$

where η is the *learning rate*, a real number between 0 and 1. Following the previous notations, the detailed procedure to calculate the gradient $\frac{\partial E}{\partial w_{ij}}$ is as follows:

1. Calculate the derivatives of the error function in Equation (5.4.21) with respect to the output units:

$$
\frac{\partial E}{\partial y_s} = y_s - z_s
$$

$$
\begin{aligned}
\frac{\partial E}{\partial net_{s,K}} &= \frac{\partial E}{\partial y_s} \frac{\partial y_s}{\partial net_{s,K}} \\
&= e_{s,K} f^{'}(net_{s,K})
\end{aligned}
\tag{5.4.23}
$$

2. Calculate the derivatives of the error function with respect to the hidden units. Work backward through hidden layers, using the chain rule. To do so, we introduce the *error signal*:

$$
\delta_j := \frac{\partial E}{\partial net_j}, \tag{5.4.24}
$$

where j is the index of an arbitrary unit in the network.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(a) Calculate the error signals of the last hidden layer:
Let j be the index of a unit in the last hidden layer which is the closest to the output layer. Since the error E depends on hidden unit j only through its connection to the output units, we have

$$\begin{aligned}\delta_{j,K-1} &= \frac{\partial E}{\partial y_{j,K-1}}\frac{\partial y_{j,K-1}}{\partial net_{j,K-1}} \\ &= e_{j,K-1}\frac{\partial y_{j,K-1}}{\partial net_{j,K-1}} \\ &= \frac{\partial y_{j,K-1}}{\partial net_{j,K-1}}\sum_{s=1}^{N_{o,K}}\frac{\partial E}{\partial net_{s,K}}\frac{\partial net_{s,K}}{\partial y_{j,K-1}}\end{aligned} \tag{5.4.25}$$

From Equation (5.4.19) and (5.4.20), and given the definition of the error signal in Equation (5.4.24), we have $\delta_{s,K} = \frac{\partial E}{\partial net_{s,K}}$, $w_{js} = \frac{\partial net_{s,K}}{\partial y_j}$, and $f^{'}(net_{j,K-1}) = \frac{\partial y_{j,K-1}}{\partial net_{j,K-1}}$. Thus,

$$\delta_{j,K-1} = f^{'}(net_{j,K-1})\sum_{s=1}^{N_{o,K}}\delta_{s,K}w_{js} \tag{5.4.26}$$

(b) Calculate the error signals for hidden layers before the last hidden layer:
The error signals for the hidden units in hidden layer k ($k < K$) are based on the calculation of the error signal of the $(k+1)$-th layer due to the chain rule. That is to say, except for the units of the last hidden layer, the error signal of each unit in other hidden unit can be computed recursively as

$$\delta_{j,k} = f^{'}(net_{j,k})\sum_{s=1}^{N_{o,k+1}}\delta_{s,k+1}w_{js} \tag{5.4.27}$$

where j and s are indicators of unit in the hidden layers k and $k+1$, respectively, before the last hidden layer.

Having computed all the error signals for all the hidden units as well as the one for the output units, the derivatives of error function with respect to the weights w_{ij}, where i is the indicator in the hidden layers $k-1$, can be written as:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_{j,k}}\frac{\partial net_{j,k}}{\partial w_{ij}} = \delta_{j,k}y_{i,k-1} \tag{5.4.28}$$

and we recover Equation (5.3.13). Then, the change of weight from unit i to unit j is given by

$$\Delta w_{ij} = -\alpha\delta_{j,k}y_{i,k-1} \tag{5.4.29}$$

Thus, introducing the concept of error signal makes the back propagation process more comprehensible, as only the error signal propagates backward through the network. The changing amount of the weight is proportional to the product of the error signal of its destination unit and the output of its source unit.

5.4.3 Summary

To summarise, the forward pass calculates the network output, while the backward pass updates the weights to minimise the error. The method used in backward pass is called back propagation, and a forward pass together with a backward pass is regarded as one *loop*. The training process continue until some *stopping criteria* is met, such as the error function is small enough, or, it stops to decrease after a certain number of loops.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 6

Recurrent neural networks

6.1 Presenting recurrent neural networks

6.1.1 An overview

Recurrent Neural Networks (RNNs) are extension of *Feedforward Neural Networks* (FNNs) discussed in Section (5.3), where we allow connections among hidden units associated with time delay (see Elman [1990]). Thus, RNNs can retain information about the past, and, as such, discover temporal correlations between events occurring far in the past. While the feedforward neural networks have no cycles for connections, a recurrent neural network has feedback connection, meaning that the nodes of the network have cyclical connections between them (see Figure 6.1a). The existence of cycles allows RNNs to develop a self-sustained temporal activation dynamics along its recurrent connection pathways, even in the absence of input, making them a dynamical system.
As shown in Figure 6.1b, RNNs can be visualised by unfolding the RNNs along the whole input sequence. That is, in the case of time series (notion of time), the RNN are unfolded through time (see Figure (6.2)). The unfolded graph has no cycles, which is the same as FNNs, so that the forward pass and backward pass of MLP can be applied (see Section (5.1.3)). Recall from Section (5.1.3.3) that the MLP is an approximator for nonlinear functions, and since RNN can be unfolded to a deep feedforward network, it has the advantage of the MLP, making it a better approximator.
There are two main classes of RNNs, (1) the first one being characterised by an energy-minimising stochastic dynamics and symmetric connections (see Taylor et al. [2007]), and (2) the second featuring a deterministic update dynamics and directed connections. We are going to concentrate on the latter.

The biggest feature of recurrent neural network is that it has a sparsely connected hidden layer called reservoir, which enables RNNs to have short term memory that captures information about what has been calculated so far. Usually, this recurrent part of the structure is essential in learning complex patterns. In fact, RNNs have proved to be an attractive form for modelling non-linearity due to their ability to approximate any dynamical system with arbitrary precision (see Siegelmann et al. [1991]). As discussed in Section (5.1.3.3), Funahashi et al. [1993] showed that under mild and general assumptions, RNNs are universal approximatiors of dynamical systems. Indeed, as an equivalent theory of universal approximation for MLPs, it is said that a single hidden layer RNN with sufficient hidden units can approximate any sequence to arbitrary accuracy (see Hammer [2000]). Thus, the RNN is a natural generalisation of feedforward neural networks to sequences (see). Some applications include motion capture data (see Sutskever [2008]), text (see Sutskever et al. [2011]), music (see Boulanger-Lewandowski et al. [2012]), among others.

For simplicity of exposition, we are going to focus on a simple RNN with one self-connected hidden layer, as shown in Figure 6.1a. As trivial the difference in topology between FNNs and RNNs may seems, the advantage of the RNNs over the FNNs is profound. The latter can only map a limited number of inputs to a limited number of outputs, while a simple RNN can map the entire time series to some outputs. This is significant for time series prediction, as the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

long history can be fed to the network, and the existence of the recursive connection allows the network to memorise information of previous time steps, which is useful considering that financial time series are mostly serially correlated.

(a) A simple RNN (b) RNN unfolded

Figure 6.1: Recurrent Neural Network

Figure 6.2: Recurrent Neural Network in time

6.1.2 The algorithm

6.1.2.1 Forward pass

Following the process described in Section (5.4) for the MLP, we now consider a sequence of length T with N_i inputs, one hidden layer ($K = 2$) with $N_{o,K-1}$ hidden units (neurons) and N_K output units. The i-th external input at time t is denoted by $x_i(t)$, and we let $net_{h,K-1}(t)$ and $y_h(t)$ be the *net value* (summation of the value received) and the output of unit h (h is the index of the neuron), respectively, in the hidden layer at time t. After unfolding the network, the forward pass of the RNN is similar to that of a MLP. The only difference being that the unit in the hidden layer receives values from both the current external inputs and the previous output of all hidden units.
Mathematically, the net value of a hidden unit is given by

$$net_{h,K-1}(t) = \sum_{i=1}^{N_i} w_{ih} x_i(t) + \sum_{h'=1}^{N_{o,K-1}} w_{h'h} y_{h'}(t-1) \tag{6.1.1}$$

where $h^{'}$ is a unit on the previous hidden layer. Then the output value of the hidden unit is calculated by applying the sigmoid activation f to the net value

$$y_h(t) = f(net_{h,K-1}(t)) \tag{6.1.2}$$

such that the output of hidden units at each time step can be computed recursively, starting from the first time step $t = 1$. Initial value of $y_h(0), \forall h$ is commonly set to be zero, meaning that the network has not received information before the beginning of forward pass. Still, some researchers tend to have nonzero intial value by which they found

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

that the network is more stable (see Zimmermann [2006]). The net value of the output unit, $net_{s,K}$, depends only on the output value of the hidden layer so that the output units are synchronised with the hidden units

$$net_{s,K}(t) = \sum_{h=1}^{N_{o,K-1}} w_{hs} y_h(t)$$

The error function is given by $E = \sum_{t=1}^{T} E_t$ where $E_t = \mathcal{L}(y_h(t))$. It can be defined as

$$E = \sum_{t=1}^{T} \sum_{s=1}^{N_K} \frac{1}{2} (y_s(t) - z_s(t))^2 \tag{6.1.3}$$

where $z_s(t)$ is the target value of the output unit s at time t. However, this is not the unique form of error function, as it should be based on the training algorithms of the RNN, which we will discuss in the next section.

6.1.2.2 Backward pass

We explained in Section (5.3) how to calculate the derivatives of the error function with respect to the weights in an MLP. In the case of RNNs, there are mainly two algorithms to calculate the weight derivatives:

1. The Real Time Recurrent Learning (RTRL) (see Robinson et al. [1987], Williams et al. [1989]).
2. The Backpropagation Through Time (BPTT) (see Werbos [1990], Williams et al [1992]).

In ordrer to describe the learning algorithm of the RNN and detail its associated problems, we focus on the BPTT, as it is simple conceptually and more efficient in computation time (see Graves [2008] [2012]). The model is expressed as a multilayer one and backpropagation is applied on the unrolled model. Figure 6.3 illustrates the scheme for updating the weights. Note that the same weights are used in every time step. The error function is defined in Equation (6.1.3), which include all the time steps.

Figure 6.3: Weights in BPTT

For MLPs, the error signals δ_j, $j = 1, 2, ...$, are computed in Equations (5.4.23), (5.4.25) and (5.4.27) according to the type of unit. However, what is common for the error signals of the hidden units is that they depend on the sum of error signals that flow back to them. Since unfolded RNNs are equivalent to MLPs, such rule can also be applied. As illustrated in Figure 6.3, we consider the hidden unit h_2 at time $t = 2$, which has two outgoing connections. We can

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

see that the error signal of hidden unit h_3 at time $t = 3$, and that of the output unit o_2 at time $t = 2$ would flow back to unit h_2 during back propagation through time. The red dashed arrows in Figure 6.4 illustrates the flow of error signals backward to a particular hidden unit. Hence, the error signal at unit h_2 can be computed as

$$\delta_{h_2} = \delta_{h_3} + \delta_{o_2}$$

Generally, the error function depends on the output of hidden layer not only through the current output layer, but also through the hidden layer next time. Thus, the error signal at unit h and time t is given by

$$\delta_h(t) = f^{'}(net_{h,K-1}(t))\Big(\sum_{s=1}^{N_K} \delta_s(t) w_{hs} + \sum_{h'=1}^{N_o} \delta_{h'}(t+1) w_{hh'}\Big) \tag{6.1.4}$$

where $\delta_j(T+1) = 0, \forall j$. Hence, starting the backward pass at the last time T, we apply Equation (6.1.4) recursively. Having computed the error signals, we can now calculate the derivative of the error function with respect to the weight via the chain rule, getting

$$\frac{\partial E}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial E}{\partial net_j(t)} \frac{\partial net_j(t)}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j(t) y_i(t) \tag{6.1.5}$$

Figure 6.4: The flow of error signals and vanishing gradient problem in back propagation. The darkness of the shading of the nodes in the unfolded recurrent network indicates the degree of influence (sensitivity) of error signal of output unit at time $t = 5$. The darker the shade represents higher sensitivity. The sensitivity decays as the error signal is back propagated through time.

6.1.3 The vanishing gradient problem

Even though recurrent neural networks (RNNs) are capable of processing serially correlated sequences, the length of sequence that a standard recurrent neural network can access is actually very limited. It results from the fact that the gradients would either decay (vanish), or, blow up (explode), when cycling around the recursive connections for too many times, which is usually referred to as *vanishing gradient problem* (see Bengio et al. [1994]). We see that:

- Vanishing gradient makes it difficult to know which direction the parameters should move to improve the loss function.
- Exploding gradient makes the learning unstable.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We can see in Equation (6.1.5) that the derivative of the error with respect to a weight depends on the error signal of the weight's destination unit, j, which itself depends on the derivative of the activation function.
Recall from Section (5.2.3), the derivative of the sigmoid function can explode or vanish depending on the slope of the function. As the number of time steps get larger and larger, so does the number of layers in the unfolded RNN. This is illustrated as follows:

- Assuming $c = 1$ in the sigmoid function, the error signal of a hidden unit at time $t = T$ would be propagated back through $(T - 1)$ layers, until a hidden layer at time $t = 1$. That is, it would be multiplied by $(T - 1)$ sigmoid activations' derivatives, all in the range $[0, \frac{1}{4}]$. For T not too large, the vanishing problem does not have much impacts on the network, but for larger T the gradient would decay exponentially.
- Similarly, for large enough c, the derivative of the sigmoid function near the origin ($x = 0$) would be larger than 1, leading to gradient explosion.

This property of sigmoid function is illustrated in Figure 6.5. For these reasons, training RNNs with standard gradient descent algorithm is only feasible for small time steps. For longer time dependencies, the gradient vanishes as the error signal is propagated back through time, so that the network weights are never adjusted correctly when taking the events far back in the past into account (see Hochreiter et al. [2001]).

Figure 6.5: Plot of $\frac{1}{1+e^{-2x}}$ and its tangent lines. The black curve is the sigmoid function ranged $[0, 1]$ with $c = 2$. The red line is the flat tangent line.

6.1.4 Dealing with long-range dependencies

6.1.4.1 Description

We saw in Section (6.1.2) that recurrent neural networks (RNNs) were powerful models, since they combine two properties:

1. Distributed hidden state allowing them to store a lot of information about the past efficiently.
2. Non-linear dynamics allowing them to update their hidden state in complicated ways.

In RNNs, we can specify the targets in several ways:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Specify desired final activities of all the units.
- Specify desired activities of all units for the last few steps: It is good for learning attractors. Further, it is easy to add in extra error derivatives as we backpropagate.
- Specify the desired activity of a subset of the units. The other units are input or hidden units.

The differences between the forward and backward passes are as follows:

- In the forward pass we use squashing functions (like the logistic) to prevent the activity vectors from exploding.
- The backward pass, is completely linear. If we double the error derivatives at the final layer, all the error derivatives will double.
- The forward pass determines the slope of the linear function used for backpropagating through each neuron.

However, the problem of exploding or vanishing gradients, described in Section (6.1.3), results in RNNs having difficulties dealing with long-range dependencies. To go around, five effective ways of learning an RNN have been proposed:

1. Long Short Term Memory: Build the RNN out of little modules that are designed to remember values for a long time.
2. Hessian Free Optimisation: Deal with the vanishing gradients problem by using a fancy optimiser that can detect directions with a tiny gradient but even smaller curvature. For example, the HF optimiser by Sutskever et al. [2011].
3. Echo State Networks: Initialise the input, the hidden states and hidden states, the hidden states and output. The hidden state has a huge reservoir of weakly coupled oscillators which can be selectively driven by the input. ESNs only need to learn the connections from the hidden states to the output.
4. Good initialisation with momentum: Initialise like in Echo State Networks, but then learn all of the connections using momentum.
5. Gradient norm clipping strategy: Dealing with the exploding gradient problem, Mikolov [2012] proposed to rescale its norm whenever it goes over a threshold.

6.1.4.2 An overview

We will detail LSTM and ESNs in the following sections, but we first describe the basic principles of the solutions introduced above.

6.1.4.2.1 The LSTM In the LSTM, Hochreiter et al. [1997] designed a memory cell by using logistic and linear units with multiplicative interactions. The cell acts as an accumulator of the state information: it is accessed, written and cleared by several self-parameterised controlling gates. The information in and out of the unit is guarded by learned input and output gates. The information gets into the cell whenever its "write" gate is on. The information stays in the cell so long as its "keep" gate is on. The information can be read from the cell by turning on its "read" gate. This over-parametrisation of the activation function preserve memory from the distant past. The main advantage of using the memory cell and gates to control the information flow is that the gradient will be trapped in the cell (this is the constant error carousels presented in Section (6.2.1)) and it will be prevented from vanishing too quickly. As an example, reading cursive handwriting is a natural task for an RNN. The input is a sequence of (x, y, p) coordinates of the tip of the pen, where p indicates whether the pen is up or down. The output is a sequence of characters. Graves et al. [2008] showed that RNNs with LSTM were producing good results for reading cursive writing. They used a sequence of small images as input rather than pen coordinates. However, as discussed in Section (6.1.3), the computational power of RNNs makes them very hard to train.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

6.1.4.2.2 Hessian Free Optimisation The Hessian-Free Optimiser is used in conjunction with structural damping. The vanishing gradient problem is addressed as follows: In high dimensional spaces there is a high probability for long term components to be orthogonal to the short term ones. It allows the Hessian to rescale these components independently. In the case of exploding gradient structural damping is an enhancement forcing the Jacobian matrix to have small norm.

6.1.4.2.3 Reservoir computing It is easier to consider reservoir computing (Echo State Networks and Liquid State Machines) with sparse reservoir. The key idea of ESNs is as follows: A simple way for learning a feedforward network is to make the early layers random and fixed them. Then, we just learn the last layer which is a linear model that uses the transformed inputs to predict the target outputs. The equivalent idea for RNNs is to fix the input to hidden connections and the hidden to hidden connections at random values and only learn the hidden to output connections. This way we do not have to learn the hidden connections. Assuming linear output units, the learning becomes very simple. However, it is important to set the random connections very carefully so the RNN does not explode or die. Setting the random connections in Echo State Network is as follows:

- Set the hidden to hidden weights so that the length of the activity vector stays about the same after each iteration. It allows the input to echo around the network for a long time.
- Use sparse connectivity (set most of the weights to zero) to create lots of loosely coupled oscillators.
- Choose the scale of the input to hidden connections very carefully. They need to drive the loosely coupled oscillators without wiping out the information from the past that they already contain.
- Since the learning is very fast, we can try many different scales for the weights and sparsenesses to improve convergence.

Since ESNs just fit a linear model, they can be trained very quickly. Further, they demonstrate that it is very important to initialise the weights sensibly. While they can do impressive modelling of one-dimensional time-series, they experience some difficulties when modelling high-dimensional data such as pre-processed speech. Given a complex task, they require more hidden units than a RNN since the latter also learn the hidden to hidden connections. In fact, Sutskever [2012] showed that RNNs can be trained very effectively if the weights are initialised using the ESN methods.

6.1.4.2.4 Gradient norm clipping strategy We discussed in Section (6.1.3) the vanishing and the exploding gradient problems when training recurrent neural networks. Mikolov [2012] proposed to clip the elements of the gradient when they exceed in absolute value a fixed threshold. See algorithm (10).

Algorithm 10 Norm clipping

Require: The vector of model parameters ψ and the threshold ξ. The gradient is denoted g.

1: $\widehat{g} \leftarrow \nabla_{\psi} g$
2: if $\|\widehat{g}\| \geqslant \xi$ then
3: $\widehat{g} \leftarrow \frac{\xi}{\|\widehat{g}\|}\widehat{g}$
4: end if

Diverging from the BPTT, Pascanu et al. [2013] followed a dynamical systems perspective (see Doya [1993]) and stated that when gradients explode we have a cliff-like structure in the error surface. Thus, similarly to Mikolov [2012], they proposed a gradient norm clipping strategy to deal with exploding gradients. They also proposed a soft constraint for the vanishing gradients problem. This regularisation term forces the Jacobian matrices to preserve the norm only in relevant directions.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

6.1.5 Mathematical representation

We now briefly present a mathematical representation of RNNs and introduce the LSTM (see Section (6.2)), the GRU, and reservoir computing (see Section (6.3)) as special cases. We consider the problem of modelling sequential data of the form $(x_1, ..., x_T) \in \mathcal{X}^T$, for some domain $\mathcal{D}$ and time length T. Being at time t, the input variable (or variable source) is denoted by x_t, the state variable is denoted by h_t, and the output is the next input value $y_t = x_{t+1}$, for every $t = 1, ..., T-1$. RNNs consider two sources of input, the present and the recent past, which are combined to determine how the network should respond to new data. Assuming a single layer, an RNN predicting the next item in the sequence can be defined in terms of a state space model (SSM) as (see Choe et al. [2017])

$$\begin{aligned} h_s &= f_\alpha(h_{s-1}, x_s), \forall s = 1, ..., t \\ x_{t+1} &= g_\beta(h_t) \end{aligned} \tag{6.1.6}$$

for some non-linear functions f_α and g_β parametrised by vector-valued parameters α and β, respectively.

In the LSTM model (see Hochreiter et al. [1997]), the main idea is to let memory cells act as accumulator of the state information and be capable of preserving gradients from earlier data. Following Graves [2013], we denote c_t the memory cell at time t which receives as input the previous cell memory c_{t-1}, the previous hidden state h_{t-1}, and the current input vector x_t. The resulting state of the cell is composed of the vectors c_t and h_t, the memory and hidden state, respectively. A LSTM cell has three gates: input, output and forget gates. Their values are computed based on linear combinations of the input vector x_t with the previous cell hidden state h_{t-1} as an input to a sigmoid activation function. We denote i_t the input gate which accumulate information when open, f_t is the forget gate, and o_t is the output gate. The single cell LSTM is illustrated in Figure (6.6).

Figure 6.6: LSTM cell.

The vector $\tilde{c}_t$ corresponds to the candidate values for the cell state that results from the combination of the input vector x_t with the previous cell hidden state h_{t-1} passed through an hyperbolic tangent activation function.
The LSTM architecture is defined as follows:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$f_{LSTM}(c_{t-1}, h_{t-1}, x_t) = [c_t; h_t]$$

where

$$\begin{aligned}
i_t &= f_\sigma\big(W_i x_t + U_i h_{t-1} + b_i\big) \\
f_t &= f_\sigma\big(W_f x_t + U_f h_{t-1} + b_f\big) \\
\tilde{c}_t &= g\big(W_{\tilde{c}} x_t + U_{\tilde{c}} h_{t-1}\big) \\
c_t &= i_t \circ \tilde{c}_t + f_t \circ c_{t-1} \\
o_t &= f_\sigma(W_o x_t + U_o h_{t-1}) \\
h_t &= o_t \circ g(c_t)
\end{aligned}$$

where f_σ is a sigmoid function and $g = \tanh$ is a hyperbolic tangent activation function. The operator $\circ$ denotes the Hadamard product [1]. Also, $\alpha = (W, U)$ for some weight matrices W and U. The forget gate decides what to keep from the previous cell memory in the component-wise product operation $(f_t \circ c_{t-1})$, while the input gate controls what candidate values are to be used in the $(i_t \circ \tilde{c}_t)$ operation, similarly to the forget gate. By combining the result of these two operations the current cell state is formed. At last, the final output from the LSTM unit, the hidden state h_t results in cell state that was previously calculated above, passed through a hyperbolic tangent activation function, and filtered by the output gate. These gating mechanisms make sure that dependencies are captured across long time ranges. This is because the cell state sums activities over time and the derivatives distribute over sums, so that the error derivatives are sent back into time.

A bi-directional LSTM (see Schuster et al [1997]) consists in two LSTM units where the first unit reads a sequence of words $w = (w_1, ..., w_N)$ in one direction, and the other LSTM unit reads the same sequence of words in the opposite direction. Afterwards, the hidden states of each LSTM are concatenated. This is written as

$$\begin{aligned}
\overrightarrow{h}_t &= f_{LSTM}(w) \\
\overleftarrow{h}_t &= f_{LSTM}(w) \\
h_t &= [\overrightarrow{h}_t, \overleftarrow{h}_t]
\end{aligned}$$

Another popular alternative is the Gated Recurrent Unit (GRU) (see Cho et al. [2014b]), which serves as a simpler and less expensive alternative to the LSTM architecture. The GRU has fewer gates than the LSTM, and does not have a memory component. This is illustrated in Figure (6.7).

[1] Given two matrices A and B, the Hadamard product is the entry-wise multiplication of two matrices, which makes sense when both matrices have the same number of rows and columns. It is defined as

$$(A \circ B)[i, j] = A[i, j]B[i, j]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 6.7: GRU.

The equations are as follows:

$$f_{GRU}(h_{t-1}, x_t) = [h_t]$$

where

$$\begin{aligned} z_t &= \mathcal{H}_\sigma\big(W_z x_t + U_z h_{t-1} + b_z\big) \\ r_t &= \mathcal{H}_\sigma\big(W_r x_t + U_r h_{t-1} + b_r\big) \\ \tilde{h}_t &= g\big(W_{\tilde{h}} x_t + U_{\tilde{h}}(r_t \circ h_{t-1}) + b_{\tilde{h}}\big) \\ h_t &= (1 - z_t) \circ \tilde{h}_t + z_t \circ h_{t-1} \end{aligned}$$

There are two gates in this setting, a reset gate r_t that controls how the previous cell memory h_{t-1} is used to compute a proposed update $\tilde{h}_t$, and an update gate z_t that decides how much of the proposed update will actually be incorporated into the old cell memory h_{t-1} to create the new cell memory h_t. If we set the reset gate to all ones, and the update gate to all zeros, we again arrive at the plain RNN model discussed above. The gating mechanism allows GRUs to better handle long-term dependencies. By learning the parameters for its gates, the network learns how its internal memory should behave, given that the gates define how much of the input and previous state vectors should be considered.

6.2 The long short-term memory

The impact of RNNs in nonlinear modelling has been limited because they are difficult to train by gradient-descent methods. The gradual change of network parameters during the learning process leads the network to bifurcations, where the gradient information degenerates and may become ill-defined (see Doya [1992]). Further, many update cycles may be necessary to obtain convergence of a few parameters, leading to long training times. In addition, when dealing with long-range memory, the gradient information may dissolves exponentially over time. One remedy is to use the Long Short-Term Memory networks (LSTM), which we are now going to describe.

6.2.1 The constant error carousel

One consequence of the vanishing, or exploding, gradient problem described in Section (6.1.3) is that the traditional RNN will fail when involved with large time series. Since the vanishing gradient problem stems from the fact that

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

the tangent line of the sigmoid function is either flat or steep, it is reasonable for some units to have constant gradient equating to 1. This kind of unit has been presented as the *Constant Error Carousel* (CEC) (see Hochreiter et al. [1997]).
In order to describe the advantageous properties of CEC, we consider a single unit, j, with a single connection to itself. According to the rule of calculation of error signals, the jth error signal satisfies

$$\delta_j(t) = f_j^{'}(net_j(t))\delta_j(t+1)w_{jj}$$

where $f_j(\cdot)$ is the activation function of unit j. A constant error flow implies that $\delta_j(t) = \delta_j(t+1)$, leading to

$$f_j^{'}(net_j(t))w_{jj} = 1$$

This is an ordinary differential equation (ODE)

$$\frac{\partial f_j(net_j(t))}{\partial net_j(t)} = \frac{1}{w_{jj}}$$

which we can integrate, getting

$$f_j(net_j(t)) = \frac{net_j(t)}{w_{jj}}$$

for arbitrary $net_j(t)$. Thus, the activation function f_j for the jth unit should be **linear**

$$y_j(t+1) = f_j(net_j(t+1))$$

Since unit j has only one connection to itself, we get

$$net_j(t+1) = w_{jj}f_j(t)$$

and from the equation of $f_j(net_j(t))$, we get

$$f_j(w_{jj}y_j(t)) = \frac{w_{jj}y_j(t)}{w_{jj}} = y_j(t)$$

We can therefore set the activation function to be the identity function $f_j(x) = x, \forall x$, obtaining $w_{jj} = 1$. The unit j above can be extended to CEC by adding some extra features. A multiplicative *input gate unit* and a multiplicative *output gate unit* are introduced to control the input and output of the CEC, respectively. The input gate unit can prevent the memory of unit j from being overwritten by irrelevant inputs, while the output gate unit avoid perturbation of other units by controlling the output of unit j. The CEC ensures that the error signal arriving at the memory cell would not be scaled up or down during back propagation, and thus can avoid exponential gradient.

6.2.2 Network architecture

A unit which include the input gate unit, the output gate unit and the CEC is called a *memory cell* (see Figure 6.8). We are now going to describe the *Long Short-Term Memory* (see Hochreiter et al. [1997]) which is a recurrent neural network consisting of set of *memory blocks*, where each block contains one or more memory cells. In fact, the LSTM network is the same as that of a standard recurrent neural network, except that the conventional hidden units are replaced by a memory block with cells and gates encapsulated in. Surely, the hidden layer of the LSTM network can be a mixture of conventional hidden units and memory blocks, but the former is not necessary. The v-th memory cell of the j-th memory block is denoted by c_j^v, the net value of the cell, the input gate and the output gate are $net_{c_j^v}$, net_{in_j} and net_{out_j}, respectively, the output value of the cell, the input gate and the output gate are $y^{c_j^v}$, y^{in_j} and y^{out_j}, respectively. Note that memory cells in the same memory block are controlled by the same input gate and output gate (see Figure 6.8).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 6.8: Architecture of a memory block with one memory cell. The CEC is a central linear unit with a self-connected weight 1.0. gate units use inputs from other units to decide whether to access or discard certain information.

As with a standard recurrent network (see Section (6.1)), we have

$$y^{in_j}(t) = f_{in_j}(net_{in_j}(t))$$

$$y^{out_j} = f_{out_j}(net_{out_j}(t))$$

where f_{in_j} and f_{out_j} are the activation function of the input gate and output gate of memory block i, and

$$\begin{aligned} net_{in_j}(t) &= \sum_u w_{in_j u} y^u(t-1) + \sum_{n=1}^{N_i} w_{in_j n} x_n(t) \\ net_{out_j}(t) &= \sum_u w_{out_j u} y^u(t-1) + \sum_{n=1}^{N_i} w_{out_j n} x_n(t) \\ net_{c_j^v}(t) &= \sum_u w_{c_j^v u} y^u(t-1) + \sum_{n=1}^{N_i} w_{c_j^v n} x_n(t) \end{aligned}$$

where the superscript u stands for memory blocks and other traditional hidden unit. The output of the memory cell is different from that of a conventional hidden layer. An additional variable $s_{c_j^v}$, called *internal state*, should be considered

$$\begin{aligned} s_{c_j^v}(t) &= s_{c_j^v}(t-1) + y^{in_j}(t) g(net_{c_j^v}(t)),\ t > 0 \\ s_{c_j^v}(0) &= 0, \end{aligned} \tag{6.2.7}$$

Actually, the first line of Equation (6.2.7) can be written as

$$s_{c_j^v}(t) = 1.0 \times s_{c_j^v}(t-1) + y^{in_j}(t) g(net_{c_j^v}(t))$$

where 1.0 corresponds to the self-recursive connection with weight 1.0 in Figure 6.8, which is essentially w_{jj} obtained by the introduction of a simple self-connected unit j without the vanishing gradient problem. So the output of the cell can be computed as

$$y^{c_j^v}(t) = y^{out_j}(t) h(s_{c_j^v}(t))$$

where g and h are the squashing function of the net value of memory cell and the internal sate $s_{c_j^v}$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 6.9: Connections in LSTM network. The example network consists of one input unit, a hidden layer of two single-cell LSTM memory blocks and one output units. Suppose the input is at time t. The the sources of the blue connections are from the previous one time step $t-1$. while the red connections convey information of current time step t. IG:input gate, OG: output gate, CEC:constant error carrousel

It is important that the gate units and the internal states are only visible within the cell, and that only the output of the cell can connect to other blocks (including gates and cell) in the hidden layer. Therefore, the net value is the summation of current input values and the output of memory cells $y^{c_i^j}$ from previous time step. Further, Graves et al. [2005] made some modifications to the original model of Schmidhuber and Hochreiter by modifying the rule connecting the units. Only the outputs memory cells are allowed to connect to other blocks, and the CECs and outputs of gate units are only visible within the memory blocks they are located in. As an example, Figure 6.9 shows the types of connections between two blocks. The gates units enable the memory cells in LSTM to store as well as to access information for a long period of time, and thus alleviate the vanishing gradient problem. For instance, if the activation function of the input gate is zero (the input gate is closed), then the memory of the cell will not be overwritten by the current input, thereby making it accessible to the network at a later time, as long as the output gate is open.
However, note that the original LSTM has a weakness. When the sequence of time is long and have not been segmented to reset the network at certain time, Equation (6.2.7) implies that the internal state will grow infinitely and the network will collapse. To remedy this problem, Gers et al. [2000] proposed to add a *forget gate* to the self-connection with weight 1.0 in the original LSTM cell (see Figure 6.8). Then the revised equation for the internal state is extended from Equation (6.2.7) as follows:

$$\begin{aligned} s_{c_j^v}(t) &= y^{\phi_j}(t) s_{c_j^v}(t-1) + y^{in_j}(t) g(net_{c_j^v}(t)),\ t > 0 \\ s_{c_j^v}(0) &= 0. \end{aligned} \tag{6.2.8}$$

where $y^{\phi_i}(t)$ is the value of the forget gate of the i-th memory block, which is squashed between zero and one. The forget gate is analogous to the reset operation of the memory cell. So far, the gate units are not entitled to control the internal state, as the gates have merely two sources of input: from the current input units and the previous output of all memory cells. In other words, the gates unit can only observe the cells' output. Once the output gate is closed, there is no way for gates to access the CEC they are supposed to control, which results in insufficient information that do harm to the performance of the network. In order to avoid the lack of information of internal states, another augmentation of the LSTM with forget gate was also introduced by Gers et al. [2002]. Weighted *peephole weights* are added in order to connect from the CEC to all gates of the same memory block. This effective remedy ensures that all the gates can "inspect" the internal state currently, even if the output gate is closed (see the dashed arrow in Figure 6.10).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

6.2.2.1 Summary

We can summarise the types of connections of the modern LSTM as follow

- Outgoing connections
  - outside memory block
    Cells' output can feed to any types of units in any blocks in the hidden layer and the output units and they are the only output that can be connected to other blocks and output units.
  - inside memory block
    The gate unit can only pass value forward to cells of the block where the gate belong to. The internal state can connect to all types of gate within the same block.
- Incoming connections
  The memory cell's input can receive outputs from the hidden layer and the input layer;
  - The gate units can receive from the outputs from the hidden layer, input layer and the value of internal states.
  - The output unit can only receive value from the outputs of conventional hidden units and memory cells.

By including the forget gate and peephole weights, the traditional LSTM evolved to the modern LSTM and the update scheme in Hochreiter et al. [1997] needs refinement.

6.2.3 The learning algorithm

Details and proofs of the learning algorithm of the LSTM can be found in Hochreiter et al. [1997]. We display the learning algorithm in Figure 6.11. We emphasise here that the update scheme involves truncated derivative which are used to enhance the efficiency of the network. Note that the *training error* is the average of the $|z_t - y_t|$, $t = 1, \cdots, T$, in Figure 6.11. The weight update scheme is illustrated in Figure 6.12. The updated algorithm for the backward pass is a combination of truncated BPTT and customised RTRL. The former refers to BPTT using truncated derivatives. To be specific, standard BPTT is applied to output units, truncated BPTT is implemented by output gates, while weights related to input gates, forget gates and memory cells use a truncated RTRL. By comparing Figure 6.3 with Figure 6.12 we can get an idea of the differences between the full BPTT and the truncated one used by LSTM. In the BPTT algorithm for standard RNN, Equation (6.1.4) shows that the information of all time steps has to be saved for the update of weight and the weights remain unchanged during the entire feedforward process. Whereas the LSTM network updates its weights at each time step during the forward pass, so that there is no need to store all values. To summarise, the BPTT with full gradient change the weight for one time by passing forward from time $t = 1$ to $t = T$ and passing backward from $t = T$ to $t = 1$ while the truncated BPTT passes forward from $t = 1$ to $t = T$ and the weights are updated for T times.

6.2.3.1 Computational complexity of LSTM

The LSTM algorithm is efficient and its computational complexity is of order $O(W)$ per time step, where W is the number of weights (see Hochreiter et al. [1997]). Compared with BPTT, the LSTM is more economical in terms of space. In fact, calculating the full gradient of the LSTM also has some advantages since it is easy to debug and can be checked by numerical approximations (see Graves et al. [2005]). With the CEC, a LSTM block may be considered as a smart network unit, compared to the conventional hidden unit, to store information for arbitrary length of time. Therefore, the LSTM is well-suited to process and predict time series when there are unknown size of long time lags between important events.
Since financial time series has long term memory, predicting volatility by LSTM network may be rather promising.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 6.10: A modern LSTM memory block with one cell. The input, output and forget gates are the analogous of write read and reset operations for the cells. The three gates are nonlinear summation units that collect outputs from outside and inside the block. The small black dots are multiplicative units by which the activation of the cell is controlled. The gate units have sigmoid function f with range $[0, 1]$, so that 0 corresponds to the gate being closed and 1 to the gate being open. The unit with red S-shaped curve is the activation function that squashes the net value of cells' input, and the unit with green S-shaped curve is the activation function of cells' output. Usually, the functions g and h are task-specific and are encouraged to have ranges different from $[0, 1]$. The input and output gates multiply the input and output of the cell, and the forget gate multiply the previous internal state. Peephole weights from internal states to gates are shown by dashed arrow. Note, except for the self-recursive connection of the CEC, connections within the blocks have fixed weights of 1.0. The only outputs from the memory block to the rest of the network is from the upper multiplicative unit.

While the traditional RNN with weight update algorithm BPTT (Back Propagation Through Time) (see Williams et al. [1990]) and RTRL (Real-time Recurrent Learning) (see Robinson et al. [1987] and Williams et al. [1992]) and the combinations of the former two (see Schmidhuber [1992]) have been proved to occur learning failure when processing sequences with only 10 time steps (see Bengio et al. [1994], Hochreiter et al. [1997], Gers et al. [2000], Hochreiter et al. HochreiterEtAl01), the LSTM can deal with 1000 time steps and even more, outperforming those traditional RNN algorithms.
In fact, except for prediction, LSTM outperforms other RNNs in numerous aspects such as the best performance in

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

```
for i = 1 to n do
    Initialize.  Initialize all weights {w_ij(1)} randomly.
    while stopping criteria not met do
        for t = 1 to T − 1 do
            Feedforward from x_t to output y_t using weights {w_ij(t)}.
            Back propagatation.  Calculate error signals based on y_t − z_t.  Take down |y_t − z_t|.
            Update weights.  {w_ij(t)} → {w_ij(t + 1)}
        end for
        Reset network except for its weights.
    end while
end for
```

Figure 6.11: Pseudo-code of LSTM. The subscripts i and j standards for any units that have connections. y_t and z_t denote the actual output and target output of the network at time t. $z_t = x_{t+1}$. n is the number of simulations.

Figure 6.12: Algorithm of weight update of LSTM. Note that the memory cell is regarded as a normal hidden unit for simplicity. Only one hidden unit is shown.The number in bracket represents the time step.Weights are updated at each time step.

speech recognition (see Graves et al. [2011]) and the ICDAR handwriting competition in 2009.

6.3 Reservoir computing

Reservoir Computing (RC) is a special RNN originating from Echo State Networks (ESNs) (see Jaeger [2001]) and Liquid State Machines (LSMs) (see Maass et al. [2002]), which assumes that supervised adaptation of all interconnection weights are not necessary, and only training a memoryless supervised readout from it is sufficient to obtain good results. Thus, it avoids the shortcomings of the gradient-descent training of RNNs (see Section (6.1.3)). RC is based on the computational seperation between a dynamic reservoir and a recurence-free readout. The former is an RNN as a nonlinear temporal expansion function, randomly created and unchanged during training. The latter

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

produces the desired output from the expansion. This is a very simple approach avoiding the problem of bifurcations encountered during the training of RNNs, and the use of complex memory cells when learning long-term dependencies, as in the LSTM described in Section (6.2). Even though several studies aimed at understanding RC and the factors affecting its performances, none have been completely satisfactory due to the complexity of the reservoir, often resulting in contradictory conclusions. Introduction to the concepts and methodologies can be found in the literature (see Lukosevicius et al. [2009], [2012]). Goudarzi et al. [2014] compared the performance of three methods, the delay line, the NARX network, and the ESN and concluded that the first two have higher memorisation capability, but fall short of the generalisation power of the latter. Thus, we are going to briefly describe the ESN.

6.3.1 Describing the Reservoir methods

Reservoir computed models are expressed in terms of state-space models comprising a set of input, output and state variables (see Appendix (23.4.3)). They are non-linear systems represented as in Equation (23.4.13).

6.3.1.1 Description

We let the time-varying input signal be an N_i-th order column vector $U(t) = [u_i(t)]$, the reservoir state is an N_x-th order column vector $X(t) = [x_j(t)]$, and the generated output is an N_o-th order column vector $Y(t) = [y_o(t)]$. A Reservoir Computer is a collection of internal nodes, whose state vector $X(t)$ evolve in discrete time t according to the nonlinear map in Equation (10.2.6) and given in the form

$$X(t+1) = (1-\alpha)X(t) + \alpha f\big(W_{res}^\top \cdot X(t) + W_{in}^\top \cdot U(t+1) + W_{fb}^\top \cdot Y(t)\big) \tag{6.3.9}$$

for some activation function $f(\cdot)$, where the leaky integrator α is a constant controlling the speed of the dynamics. The input weight matrix is an $N_i \times N_x$ matrix $W^{in} = [w_{ij}^{in}]$ where w_{ij}^{in} is the weight of the connection from input node i to reservoir node j. The connection weights inside the reservoir are represented by an $N_x \times N_x$ matrix $W_{res} = [w_{jk}^{res}]$ where w_{jk}^{res} is weight from node j to node k in the reservoir. In presence of a bias, the output matrix is an $(N_x+1)\times N_o$ matrix $W_{out} = [w_{ko}^{out}]$, where w_{ko}^{out} is the weight of the connection from the reservoir node k to the output node o. In the case where the output nodes are also connected to the input nodes and to themselves the size of the matrix becomes $(N_x + N_i + N_o) \times N_o$. The feedback matrix is an $N_x \times N_o$ matrix $W_{fb} = [w_{ko}^{fb}]$, where w_{ko}^{fb} is the weight of the connection from the reservoir node k to the output node o. If no output feedback is needed, then W_{fb} is null. The generated output is given by

$$Y(t) = f_{out}(W_{out}^\top \cdot X(t)) \tag{6.3.10}$$

where $f_{out}(\cdot)$ is an output activation function, typically the identity or a sigmoid. The output weights are trained to minimise the squared output error

$$E = ||Y(t) - \hat{Y}(t)||^2$$

given the target output $\hat{Y}(t)$. Once we know the optimum weights of the network, we can use the model to perform a forecast. To do so, we let T be a network state update, where

$$X(t+h) = T(X(t), \overline{U}^h)$$

denote the network state resulting from an iterated application of Equation (6.3.9) when the input sequence $\overline{U}^h = U(t+1), ..., U(t+h)$ is fed to the network being in state $X(t)$ at time t.
The basic steps to construct an ESN are as follows:

- $\mathbf{W}_{in}$ and $\mathbf{W}_{res}$ are randomly initialised, an appropriate α is selected.
- For each t,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

  - Import input vector $\mathbf{U}(t)$ of the training set to the system and evolve the corresponding reservoir state $\mathbf{X}(t)$.
- A supervised learning algorithm is built on pairs of $\mathbf{X}(t)$ and $\widehat{\mathbf{Y}}(t)$.
- For each t,
  - Apply trained $\mathbf{W}_{out}$ on input vector $\mathbf{U}(t)$ of test set to compute predictions $\mathbf{Y}(t)$.

6.3.1.2 The parameters

We present some of the major characteristics one has to consider when defining a relevant model (see Lukosevicius et al. [2009] and Lukosevicius [2012]). There are several features one has to consider when designing RC:

1. Connectivity: $\{W_{in}, W_{res}, W_{fb}\}$ is generated randomly. The input, bias, reservoir and feedback scaling (respectively named s_{in}, s_{bias}, s_{res} and s_{fb}) will be considered as micro parameters. For the choice of topology, the authors suggested a permutation matrix with a medium number and different lengths of connected cycles, or a general orthogonal matrix.
   (a) N_x: The principle is that the bigger the reservoir, the better the obtainable performance, provided appropriate regularisation measures are taken against over-fitting. As ESNs are computationally cheap, we can use as big a reservoir as we can afford computationally. N_x will be considered as a macro parameter.
   (b) Sparsity: It enables fast reservoir updates. One rule of thumb is to connect at most each node to 10 other nodes in the reservoir.
   (c) Distribution: Uniform or Normal. W_{in}, W_{res} and W_{fb} have usually the same distribution. In general, the seed of the pseudo-random generator is fixed.
2. Leaking Rate α: The leaking rate can be seen as the time interval between two discrete realisations. In general, we expect its value to be close to 1.
3. Echo State Property: The state of the reservoir $X(t)$ should be uniquely defined by the fading history of the input $U(t)$ [...] The spectral radius should be greater in tasks requiring longer memory of the input. Finding clear characteristics remain a large debate among practitioners and researchers.

From these features, one can see that many meta parameters are involved, so that it is necessary to perform an additional optimisation aiming at minimising the RMSE on the validation sample. It is known as meta-optimisation (see Pedersen et al. [2008a]). The idea is to have an optimisation method act as an overlaying meta-optimiser, trying to find the best performing behavioural parameters for another optimisation method (see Pedersen et al. [2008a]). Note, parameter tuning via meta-optimiser is done in an offline manner, while adaptation of DE parameters is done online during optimisation.
According to Lukosevicius et al. [2009], the investigation shows that the error surfaces in the combined global parameter and W_{out} spaces may have very high curvature and multiple local minima. Thus, gradient descent methods are not always practical. In general, optimising reservoir is quite a hard task as we have many parameters involved. Nevertheless, the computationally cheap property of Reservoir Computing leads to the following idea: *"generate k reservoirs and pick the best."*.

6.3.1.3 The readout

Echo State Networks (ENSs) assume that if a random RNN possesses certain algebriac properties, it suffises to train a linear readout from it to obtain good performances. Even though there are several readout methods, the most popular one is the linear regression where W_{out} solve a system of linear equations. From Remark (**??**), we get

$$W_{out}^{\top} X_m = \widehat{Y}_m$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where both the matrix $X_m \in \mathbb{R}^{N_x \times T}$ and the matrix $\hat{Y}_m \in \mathbb{R}^{N_o \times T}$ [2], over the training period $t = 1, ..., T$, have a column for every training time step t. The output weights can be estimated with direct pseudo-inverse calculations, or they can be estimated with the ordinary linear regression (Wiener-Hopf solution). Thus, we get

$$W_{out}^{\top} X_m \cdot X_m^{\top} = \hat{Y}_m \cdot X_m^{\top}$$

and the weight matrix becomes

$$W_{out}^{\top} = \hat{Y}_m \cdot X_m^{\top} (X_m \cdot X_m^{\top})^{-1} = PR^{-1}$$

where $R = X_m \cdot X_m^{\top}$ is the correlation matrix of the reservoir states, and $P = \hat{Y}_m \cdot X_m^{\top}$ is the cross-correlation matrix between the states and the desired outputs. Note, $P \in \mathbb{R}^{N_o \times N_x}$ and $R \in \mathbb{R}^{N_x \times N_x}$ do not depend on the training length T and can be calculated incrementally. When R is ill-conditioned the method is numerically unstable, but computing the Moore-Penrose pseudo-inverse R^+ instead of R can improve the solution. Alternatively, we can decompose the matrix R into two triangular matrices with Cholesky or the LU decomposition (see Press et al. [1992]) and solve

$$W_{out}^{\top} R = P$$

by two steps of substitution to get $W_{out}^{\top} = PR^{-1}$.
Evolutionary search can also be used for training the linear readouts. There exists several bias to reduce the error in the validation set. One can smooth the model with regularisation functions with the Ridge regression

$$W_{out}^{\top} = P(R + \beta^2 I)^{-1} \tag{6.3.11}$$

where β controls the smoothing effect, and $I \in \mathbb{R}^{N_x \times N_x}$ is the identity matrix.

The following algorithm sums up the steps involved in the training of the ESN framework.

Algorithm 11 RC offline Training

Require: get a linear system solver (we use Cholesky Decomposition).
Require: set t_0 as a wash-out size.
Require: set β as a regularisation parameter.
1: $Y \leftarrow [y(t_0) \| \ldots \| y(T)]$
2: $Y \leftarrow f_{out}^{-1}(Y)$
3: $X \leftarrow [1 \| U(t_0) \| \ldots \| U(T) \| X(t_0) \| \ldots \| X(T)]$
4: $W_{out} \leftarrow Y \cdot X^T \cdot (X \cdot X^T + \beta^2 I)^{-1}$
5: **return** W_{out}

Note, we can also add noise $\epsilon(t)$ (sampled from uniform or Gaussian distribution) to the reservoir states

$$X(t+1) = (1-\alpha)X(t) + \alpha f\big(W_{res}^{\top} \cdot X(t) + W_{in}^{\top} \cdot U(t+1) + W_{fb}^{\top} \cdot Y(t)\big) + \epsilon(t)$$

to stabilise solutions in the model (see Jaeger [2007]). We can further adjust global control parameters to make the echo state network dynamically similar to the system we model. For instance, we can use fully connected reservoirs or sparsely connected ones.

[2] These matrices are transposed compared to the conventional notation.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

6.3.1.4 Some comments

In order to produce a reservoir with a rich enough set of dynamics the number of internal connections N_x should be large, the weight matrix W should be sparse, and the weights of the connections should be generated randomly from a uniform distribution symmetric around the zero value. Further, the network should have the echo state property (ESP), which relates asymptotic properties of the excited reservoir dynamics to the driving signal (see Jaeger [2001]). It states that the effect of a previous state $X(t)$ and a previous input $U(t)$ on a future state $X(t+h)$ should vanish gradually as $h \to \infty$, and not persist or be amplified. In reservoirs using the $tanh$ squashing function, and for zero input, the reservoir weight matrix W^{res} must be scaled so that its spectral radius [3] $\rho(W^{res})$ satisfies $\rho(W^{res}) < 1$. For any kind of inputs (including zero) and state vectors, we require $\sigma_{max}(W^{res}) < 1$ where $\sigma_{max}(W^{res})$ is the largest singular value of W^{res}. If the input comes from a stationary source, the ESP holds with probability 1 or 0. Due to the auto-feedback nature of RNNs, the reservoir states $X(t)$ reflect traces of the past input history, which can be seen as a dynamical short-term memory. Assuming a single input ESN, the short-term capacity is given by

$$C = \sum_i r^2(U(t-i), Y_i(t))$$

where $r^2(\cdot,\cdot)$ is the squared correlation coefficient between the input signal delayed by i and an output signal $Y_i(t)$ trained to memorise $U(t-i)$ on the input signal $U(t)$. For an i.i.d. input, the memory capacity C of an echo state network of size N is bounded by N. Thus, we can not train ESN on tasks requiring unbounded-time memory.

6.3.2 Choosing the parameters

Choosing the right parameters can be more beneficial than choosing the right model itself. However, it is never trivial to tune parameters of ESN. Experience will play an important role here. It is inevitable and necessary to use manual selection to get a sense of which parameters affect results more than others do, and these manually selected values could be used as the initial values for further automated selection. During the process, ensure that *only* one parameter is changed at a time. The quality of a particular parameter setting is evaluated by using root mean square error of training set and test set. Note that all the parameters ought to be optimised under a condition where adequate regularisation measure is taken to alleviate overfitting.
Practically, the input scaling, leak rate and spectral radius of reservoir weight matrix are the three most important parameters of ESN.

6.3.2.1 Input Scaling

Input weight matrix $\mathbf{W}_{in}$ is usually dense without zero elements. The entries of the matrix usually follow a uniform distribution in a symmetric range $[-a, a]$ or classic range $[0, 1]$. Often candidate values for a is 1, 0.1, 0.01. The distribution can be Gaussian or a customised one as well, whereas the boundness is not guaranteed.
The first column of the input weight matrix is usually the bias column, which corresponds to bias element $'1'$ in input vector $\mathbf{U}(t)$. Hence it does not share the same status as the rest of the columns. Thus, to reduce degree of freedom, all columns except the first column of the input weight matrix are randomized using one parameter a, and the first column uses another parameter b.
Small values of a will map input to a small range around zero, where $tanh(\cdot)$ is virtually linear, hence this setting is suitable for solving linear systems. On the other hand, large values of a will map $\mathbf{u}(n)$ to extreme values of $tanh(\cdot)$, namely -1 or 1, creating a non-linear, binary manner. Overall, input scaling controls non linearity of $\mathbf{X}(t)$.

[3] the largest absolute eigenvalue

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 6.13: activation function tanh

Figure 6.14: activation function sigmoid

The reservoir weight matrix $\mathbf{W}_{res}$ represents a structure of recurrently connected units. Similar to the case of $\mathbf{W}_{in}$, non-zero elements of $\mathbf{W}_{res}$ usually have uniform distribution in a symmetric range. However, unlike $\mathbf{W}_{in}$, $\mathbf{W}_{res}$ is sparse and has other distinct properties.

6.3.2.2 Size of the reservoir

Intuitively, larger the reservoir, the better the attainable performance. the size can be as big as 20, 50, 100, 1000, even 10000 is not uncommon. There is few condition when reservoir may be oversized: one is that the task itself is pretty much trivial, and the other could be that the size approaches meaningful fraction of number of data points, or in other words, lack of data points. Normally, if sample size is n, it is required that (see Jaeger [2002])

$$N_x <= n/2; \tag{6.3.12}$$

Typically, it also holds that $1 + N_u + N_x << n$, this constraint also reduces the risk of overfitting. At the same time, the size of reservoir should at least be the number of independent time steps that are supposed to be memorised to address the problem.

6.3.2.3 Sparsity

Though the reservoir is big, the reservoir weight matrix is always set to be *sparse* in many publications. In general, higher density tends to give higher prediction performance. However, at the same time, lower sparsity means more recurrent connections, and large non-zero matrix operations increase computational costs. Generally, above a certain threshold, connectivity barely affects the final performance. Hence, one can recommend 30% connectivity, which is robust enough to be a balance between efficiency and efficacy. Compared to others, sparsity has relatively low priority to be optimised. In Matlab, there is a special representation for sparse matrix that could speed up computations.

6.3.2.4 Spectral Radius

Being the central parameter of ESN, spectral radius is the maximum of absolute eigenvalues of reservoir matrix $\mathbf{W}$. Intuitively, it determines the width of the distribution of non-zero entries. If it is set too high, then the reservoir may hold periodic, even chaotic spontaneous attractor modes which deactivates echo state property. It has been proved that, in most cases, $r(\mathbf{W}) < 1$ ensures the echo state property (see Lukosevicius [2012]). However, it is also true that $\mathbf{W}$ with spectral radius greater than 1 retains echo state property. As a rule of thumb, if the task requires extensive input history, then it is recommended to set the spectral radius higher. The following steps are usually taken when specific spectral radius is assigned to reservoir weight matrix:

1. Initialize a reservoir weight matrix $\mathbf{W}_0$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. Normalize $\mathbf{W}_0$ to $\mathbf{W}_1$ using $\mathbf{W}_1 = \mathbf{W}_0/r_1$, where r_1 is the spectral radius of $\mathbf{W}_0$. Now that $\mathbf{W}_1$ has unitary spectral radius.

3. Multiply $\mathbf{W}_1$ by r element-wise so that it has spectral radius r.

6.3.2.5 Leak Rate

Leak rate is a real value between 0 and 1. Echo state network without leak rate α is called standard echo state network, the one with leak rate or leaky integrator is called LI-ESN. In this latter case, α can either be placed before application of $f(\cdot)$ or after. Introduction of leak rate makes it possible to learn very slow dynamic systems and replay them at various speeds (see Lukosevicius et al. [2009]).
Small values of leak rate may induce slow dynamics of $\mathbf{X}(t)$ and extend short term memory (see Jaeger [2012]). Virtually, the prediction performance is sensitive to leak rate since a small change in leak rate may lead to a big change in final result. Hence, this parameter also has relatively high priority. Some low-prioritised parameters could be assigned default values for convenience.

6.3.3 Some improvements

Separating the reservoir from the readout training allows for two research directions to be pursued independently,

1. the generation of the reservoir, and

2. the output training.

There is no reasons why the reservoir should be randomly generated and alternative methods could be used to obtain optimal reservoir design. However, no single type of reservoir can be optimal for all types of problems (no free lunch principle). Several methods have been proposed for generating the reservoir (see Lukosevicius et al. [2009]), which can be classified as

1. generic methods for generating $RNNs$ with different neuron models, connectivity patterns and dynamics.

2. unsupervised adaptation of the reservoir based on the input data $U(t)$ but not the target value $\hat{Y}(t)$.

3. supervised learning, adaptation of the reservoir using task-specific information from both $U(t)$ and $\hat{Y}(t)$.

In order to deal with different time scales simultaneously, one can divide the reservoir into decoupled sub-reservoirs and introduce inhibitory connections among all the sub-reservoirs. However, the inhibitory connections should be heuristically computed from the rest of W and W_{fb} such that they predict the activations of the sub-reservoirs one time step ahead (see Xu et al. [2007]). Alternatively, the Evolvino transfers the idea of ESNs to a LSTM type of RNNs where the LSTM RNN used for its reservoir consists of specific small memory-holding modules. In that model, the weights of the reservoir are trained using evolutionary methods (see Schmidhuber et al. [2007]). Further, ESNs like any other RNNs has only a single layer of neurons (see Figure 6.1a), making it unsuitable for some types of problems requiering multilayers. A solution is to use Layered ESNs (see Lukosevicius [2007]), where part of the reservoir connections are instantaneous, and the rest takes one time step for the signals to propagate as in normal ESNs. One can also add leaky integrator neurons to ESNs, getting leaky integrator ESNs (Li-ESNs) performing at least as well as the simple ESN (see Lukosevicius et al. [2006]). Since the parameters a and Δt control the speed of the reservoir dynamics, small values result in reservoirs reacting slowly to the input. Note, depending on the speed at which the input $U(t)$ changes, we can vary Δt on-the-fly, getting a warping invariant ESNs (TWIESNs). Since checking the performance of a resulting ESN is relatively inexpensive, evolutionary methods for pre-training the reservoir developed (see Ishii et al. [2004]). Generally, one separate the topology and weight sizes of W_{res} to reduce the search space (see Bush et al. [2005]). Jiang et al. [2008] showed that by only adapting the slopes of the reservoir unit activation functions $f(\cdot)$ with an evolutionary algorithm, and having W_{out} random and fixed, a very

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

good prediction performance of an ESN could be achieved. Note, evolutionary algorithms can also be used to train the readouts. One can increase the expressiveness of the ESN by having k linear readouts trained and an online switching mechanism among them, or by averaging outputs over several instances of ESNs (see Bush et al. [2006]).

6.3.4 Empirical results

It is generally argued that ESN is expert in fitting and predicting chaotic time series. This could be verified by assessing its performance on the classic Mackey-Glass series (see Figure (6.15)). We also verify performance of the models on denoised financial time series. The modelling is under the assumption that historical price series contains all necessary information to be used to predict next day prices and returns. We let the Linear Regression (LR) model be the benchmark (see Section (12.1.1.3)). We reproduce below some results on the Mackey-Glass signal obtained by a student of ours who did his MSc thesis at QFL (see Wang [2017]).

Figure 6.15: Samples of Mackey-Glass Chaotic Time Series

In a publication by Jaeger, it is claimed that advanced ESN could yield root mean square error (RMSE) of order 10^{-7} on predicting Mackey series (see Jaeger et al. [2004]).
We try to reproduce the experiment, using 2000 data points for training, with the first one hundred reservoir states $\mathbf{X}(t)$ being discarded due to initial transient effect, and 500 data points are used for testing. Both linear regression (LR) and ESN will be implemented. For ESN, parameters are specified as follows:

Leak Rate α	0.9	Spectral Radius	1.25
Input Weight Matrix $\mathbf{W}_{in}$	$[-0.5, 0.5]$	Reservoir Weight Matrix $\mathbf{W}_{res}$	$[-0.5, 0.5]$
Reservoir Size $\mathbf{N}_x$	400	Normalization Range	$[-1, 1]$
Reservoir Density	0.3	Regularization term β	10^{-8}

After plugging all the parameters into ESN, following errors are summarised:

	Training error(RMS)	Testing error(RMS)
Linear Regression	0.0315	0.032
ESN	7.9×10^{-6}	8.0×10^{-6}

Apparently ESN produces negligible errors compared to that of linear regression. In fact, even if reservoir size N_x is reduced to 35, training and testing errors are still less than 0.001, which is less than 3% of the errors made by linear regression.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

For illustration, the predicting performance of ESN with $N_x = 35$, along with that of linear regression, are displayed below:

Figure 6.16: Mackey Series test set

In both figures, one could not virtually discern between predicted values and actual values, since the two curves nearly coincide with each other most of the time. Judging from the plot, it seems that linear regression has reasonably well fitted the data even if it has 'huge' test error compared with ESN. However, if we study more carefully the two figures, we can observe differences:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 6.17: Zoomed-in Figure Comparison

One may notice that the prediction curve of the linear regression is just a forward shift of the target curves. In other words, the prediction curve is just a *time-lagged* copy of the actual target curve if horizontal axis is discrete time. Therefore, in this case, the prediction capability of linear regression is relatively poor. On the other hand, the ESN prediction curve tends to move *in phase* with the target curve. Even at the turning point around $t = 290$, where the two curves can visually be distinguished, the trends of prediction values keep pace with that of target curves. Hence, statistically, ESN has much better test error than linear regression. Below are some samples of reservoir states $\mathbf{X}(t)$:

Figure 6.18: Evolutions of reservoir activations

In the figure above, reservoir activations actually means a vertical concatenation of bias factor, input vector and actual reservoir state, that is, $(1, \mathbf{U}(t), \mathbf{X}(t))^\top$. It is a vector of size $1 + N_u + N_x$, since N_x is large, the first 20 entries of the vector are plotted. Hence in the graph there are a total of 20 curves, with each curve being the evolutions of a fixed position within the vector $(1, \mathbf{U}(t), \mathbf{X}(t))^\top$. At the same time, the x-axis indicates $t = 1$ to 200, and the number of evolutions is 200. In the figure, the reservoir state is not saturated since not all curves converges to the same number(for example 1 or -1). Note also that the upper most curve, the bias factor, is a straight line which always equals to one.

The following is a bar plot of output weight matrix $\mathbf{W}_{out}$:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 6.19: Evolutions of reservoir states

Assuming $N_y = 1$, $\mathbf{W}_{out}$ is a vector of size $N_x \times 1$. In the figure above, $N_x = 1000$, and almost all the elements of $\mathbf{W}_{out}$ are within the range $[-3, 3]$. Hence, there are no extreme large values in $\mathbf{W}_{out}$ and the model is less likely to suffer from overfitting.

Recall that the word 'chaotic' means that a time series consists of irregular patterns, so that visually no obvious trends could be easily perceived, while 'noisy' emphasises on stochastic or irrelevant factors that may interfere or even obscure useful inputs. A time series could be both chaotic and smooth at the same time, such as Mackey Series, while noisy time series are usually filled with *wavelets* (see Appendix (23.3)).

After adding some Gaussian noise to the standard Mackey series, the following figure is plotted:

Figure 6.20: Mackey series

Figure 6.21: noisy Mackey series

As in Figure (6.21), the noisy curves are usually oscillating more frequently and therefore are much more difficult for any models to track its path. As a matter of fact, it could be seen that ESN possesses advantages in fitting and predicting chaotic time series. However, experiments show that the performance of ESN will be discounted coping with noisy time series directly.

Unfortunately, financial time series is often chaotic and noisy. In addition, great caution is required when financial data are used for backtesting. There is no doubt that flawed data can lead to misleading results. In worst cases, historical returns may be overestimated. Following are several facts about financial data:

- Dataset with high resolution offers more details, at the expense of being noisier (Aamodt, 2015, S. 50)

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Stock prices need to be adjusted after share split or dividend pay-out.
- Most importantly, many accessible datasets have survisorship bias (see Chan [2009]) since historical database does not hold information about companies which disappeared due to delistings, bankruptcies or mergers.

Along with the first two bullet points, the size of dataset also matters in a machine learning model, while the third bullet point, as well as transaction cost, is crucial in terms of backtesting.

Due to limited public access to all types of financial datasets, the focus is on modelling stock returns. Hence, regardless of survivorship bias, only survived stocks, especially those from S&P 500, are chosen for experiments. Generally, these stocks have some advantages over other stocks in terms of market cap and liquidity. The index S&P 500 itself is often used as the weather report of financial industry.

In our experiment, 100 constituents of S&P 500 are imported. For each stock, the daily closing price data are downloaded. 1200 data points are used for training (the first 100 points are used to evolve reservoir states $\mathbf{X}(t)$ and will be discarded), while another 200 points are left out for testing.

Applying linear regression to the raw log-returns of symbol MMM gives the following figures:

Figure 6.22: Linear Regression on raw stock returns

The figure shows that the log-return time series is so intractable and noisy that only approximately a straight line around zero would minimise the least square error for linear regression, where the regularisation term β is the only parameter of ridge regression. However, regardless of what β is, it always gives similar results. Here, the ratio of predicted values volatility to target values volatility would be close to zero, which is undesirable.

Now, the same raw data are fed into the ESN:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 6.23: ESN on raw stock returns

Statistically, the ESN has test error of only 5% lower than that of linear regression, despite the fact that visually their figures differ a lot. Moreover, in dealing with unprocessed data, adjustment of parameter values does not improve much the performance of the ESN.

Now, the log-return series of MMM is denoised before fed into linear regression and ESN. Choosing the regularisation term $\beta = 10^{-8}$, the linear regression gives the following error:

	Training error(RMS)	Test error(RMS)
Linear Regression	0.00472	0.0036

At the same time, the ESN is equipped with common parameter settings below,

Leak Rate α	1	Spectral Radius	0.8
Input Weight Matrix $\mathbf{W}_{in}$	$[-2, 2]$	Reservoir Weight Matrix $\mathbf{W}_{res}$	$[-0.5, 0.5]$
Reservoir Density	0.5	Regularisation Term β	10^{-8}

In Matlab, the same random initialisations for $\mathbf{W}_{in}$ and $\mathbf{W}_{res}$ are reproducible by setting values for seed. For example, rand('seed', 34);

Now, fix these parameters, and only vary the reservoir size N_x since it can sensitively affect performance of ESN. The results are displayed in triplets, which stands for N_x, training error and test error respectively.

ESN	f=tanh	f=sigmoid
No normalisation	$850, 0.00227, 0.00157$	$200, 0.00235, 0.00204$
Normalised to $[-1, 1]$	$170, 0.00209, 0.00150$	$600, 0.00228, 0.00140$
Normalised to $[-2, 2]$	$140, 0.00234, 0.00166$	$80, 0.00232, 0.00139$
Normalised to $[0, 1]$	$190, 0.00193, 0.00136$	$650, 0.00250, 0.00155$

From the table, it could be seen that:

- Normalisation generally improves task performance of ESN (note that normalisation does not virtually change performance of linear regression).
- Compared to tanh, sigmoid function works better in symmetric ranges. Especially with the range $[-2, 2]$, ESN with sigmoid activation function uses small reservoir size $N_x = 80$ and still perform better than that of tanh with a larger reservoir size $N_x = 140$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- The best test error comes from ESN with normalisation range $[0, 1]$, it reduces 62% of the test error made by linear regression. Moreover, it is also coupled with best training error in the table.

Cautiously note that these are observations from the test on stock MMM. Further experiments on large set of stocks are required to check if the above observations can be generalised.

The corresponding figures for ESN with normalisation range $[0, 1]$ and linear regression are displayed below:

Figure 6.24: ESN on denoised stock returns test set

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 6.25: ESN on denoised stock returns training set

From these figures it can be seen that ESN restores some of its prediction capability when dealing with denoised chaotic time series instead of raw time series. ESN apparently has better fittings in comparison of linear regression. Statistically, with normalisation, ESN reduces at least 57% of the test error made by linear regression.
Moreover, note that parameters are manually tuned one at a time. Therefore, if a systematic way of tuning (that is, grid search, genetic algorithm) is used, the errors are likely to be further reduced by a meaningful percentage. Moreover, optimal parameters settings are specific and unique for each stock. If the same setting is used for various stocks, the corresponding errors would be inevitably increased. Nevertheless, an experiment is done with the following parameter setting:

Leak Rate α	1	Spectral Radius	0.8
Input Weight Matrix $\mathbf{W}_{in}$	$[-2, 2]$	Reservoir Weight Matrix $\mathbf{W}_{ref}$	$[-0.5, 0.5]$
Reservoir Density	0.5	Reservoir Size N_x	350

Meanwhile, a relative large regularisation term $\beta = 0.01$ is chosen to reduce chance of overfitting. Also, the sigmoid function with normalisation range $[-2, 2]$ is used. ESN with this parameter setting is applied on 100 stocks from S&P 500 and then gives the error table as follows:

	Average Training error(RMS)	Average Test error(RMS)
Linear Regression	0.00809	0.00799
ESN	0.00498	0.00515

This time the table shows that ESN reduces the error by only 36% instead of 57% of the test error made by linear regression. This is due to the use of common parameter setting for all 100 stocks for convenience. Nevertheless, 36% reduction still reveals competitive prediction capability of ESN across stocks with a common parameter setting.

6.4 Other networks

6.4.1 Variable weights neural networks

In conventional neural networks, connection weights are fixed. That is, the characteristics of the neural network is invariant (the invariant type neural network). The adaptive ability of neural networks to an unknown environment depends on its generalisation capability. However, there exists a limit of generalisation ability in invariant type neural networks. Yasuda et al. [2006] proposed the neural network with variable connection weights, which changes its connection weights and its characteristic according to the changing environment. Lam et al. [2014] presented the variable weight neural network (VWNN), allowing its weights to be changed in operation according to the characteristic of the network inputs.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

6.4.1.1 Description

We are now going to briefly describe the VWNN. It consists of two traditional neural networks, namely tuned and tuning neural networks. The former is the one which actually classifies the input data, while the latter provides the weights to the tuned neural network according to the characteristic of the input data. The VWNN works on the principle that the connection weights are function to the external input signal.
As an example, we consider a three-layer feed-forward fully-connected neural network (denoted fixed model) with n_{in} inputs and n_{out} outputs, weights $\omega_{ji}^{(k)}$ and bias terms $b_j^{(k)}$, $k = 1, 2$. The input vector is $x(t) = [x_1(t), ..., x_{n_{in}}(t)]$ and the output vector is $y(t) = [y_1(t), ..., y_{n_{out}}(t)]$ (see Bryson et al. [1969]). We are going to modify that network to obtain its associated VWNN. To do so, we need the tuning neural networks NN_k, $k = 1, 2$, which provide connection weights $\omega_{ji}^{(k)}$, and bias terms $b_j^{(k)}$ to the fixed model. The tuning networks rely on the modified input vector $x^{'}(t)$, which consists of some selected features from $x(t)$. Given a predetermined constant selection matrix $S \in \mathbb{R}^{n^{'}_{n_{in}} \times n_{n_{in}}}$, then

$$x^{'}_{n_{in}}(t) = S \cdot x_{n_{in}}(t)$$

where $n^{'}_{n_{in}} \leqslant n_{n_{in}}$. For example

$$x^{'}_{n_{in}}(t) = \begin{bmatrix} x^{'}_1(t) \\ x^{'}_2(t) \end{bmatrix}, x_{n_{in}}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} \text{ and } S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

such that $x^{'}_{n_{in}}(t)$ selects $x_1(t)$ and $x_3(t)$ as the input of the tuning neural networks NN_k. The latter will produce output weight vector $W(t)$ consisting of all connection weights of the tuned neural network (fixed model). Eventually, the tuned neural network will then use $W(t)$ to process the input $x(t)$ and produce the output $y(t)$. See Figure (6.26) and Figure (6.27).

6.4.1.2 Application to finance

Clearly, the role of the matrix S is to reduce the dimensionality of the input signal x in view of generating dynamical weights as a function of the latter.
The method was applied to large size neural network. In the case of financial time series, we can let the matrix S be a lowpass filter or other denoising technique, so that the modified input $x^{'}$ of the tuning network becomes the filtered time series. This way, the fixed weights of the tuning network are associated with a smooth signal and the variable weights of the tuned network become a function of the filtered market returns.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 6.26: A three-layer variable-weight neural network

Figure 6.27: A block diagram of variable-weight neural network

6.4.2 Time-varying feedforward networks

We saw in Section (6.1.3) that backpropagating gradients from time step T to time step 0 could require infeasibly large amounts of memory so that one solution was to truncate the model and only backpropagates gradient k times steps. However, several recent studies showed that feedforward networks could match the results of the best recurrent models on benchmark sequence tasks such as:

- Language modelling: The goal is to predict the next word in a document given all of the previous words.
- Machine translation: The goal is to map sequences of English words to sequences of, say, French words.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Speech synthesis: One seeks to generate a realistic human speech signal.
- Other applications such as music generation.

Examples of feedforward networks proposed are the Autoregressive models and Transformer models. Rather than making predictions from a state that depends on the entire history, an autoregressive model directly predicts the output y_t using only the k most recent inputs $x_{t-k+1}, ..., x_t$. However, it relies on a strong conditional independence assumption. That is, a feedforward model assumes the target only depends on the k most recent inputs. Thus, it cannot capture patterns that extend more than k steps. Note, the is not exactly a feedforward model since it does not make the k step conditional independence assumption. However, Dauphin et al. [2017] stated that we did not need a large amount of context to do well on the prediction task on average. They showed that the Gated-Convolutional Language Model was a feed-forward autoregressive models competitive with large LSTM baseline models. Bai et al. [2018] used the technique of dilated-convolutions to make k large enough in the case of synthetic copying tasks and music generation. The main argument is that we do not need infinite memory when performing tasks related to language. Further, while feedforward models are less expressive than RNN, they benefit from parallelisation, trainability, and inference speed. As a result, Miller [2018] argued that DNN models such as autoregressive Wavenet and Transformer were replacing RNNs on a diverse set of tasks. Miller et al. [2019] proved that stable recurrent neural networks (meaning the gradients can not explode) were well approximated by feedforward networks for the purpose of both inference and training by gradient descent. However, feedforward networks make a strong conditional independence assumption. This is not the case of market returns which are multifractal. Further, the origin of this discussion derived from the vanishing and the exploding gradient problems when training recurrent neural networks (Section (6.1.3)). The problem disappear when considering Reservoir Computing (RC). Thus, it would make sense to use RC when forecasting market returns.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 7

Towards dynamically stable reservoirs

7.1 A specific implementation

7.1.1 Overview of the topology

We let $\mathbb{T} := [1, N_T]$ be a discrete time set and define:

- $u \in \mathbb{R}^{N_T \times N_U}$ the input signal at time t and we define $U := [1, N_U]$.
- $X := (x^k)_{k \in N_K} \in \mathbb{R}^{N_K \times N_T \times N_{X_K}}$ a sequence of reservoir neuron activations (often called kernels). $\forall k \in \mathbb{K} := [1, N_K]$, x_k is a factor which can be seen as a fraction of the reservoir. Factors are usually defined with the same matrix design and can be correlated or not (in terms of dimension and jump). By allowing this third dimension, we can represent several high dimensional signal in a same reservoir and in a dependent or independent manner. They all have their own set of Equations (6.3.9) (6.3.10) and parameters coming with it. We also define $x = [x^1 \| \dots \| x^K]$.
- $y, \ \hat{y} \in \mathbb{R}^{N_T \times N_Y}$ are respectively the output and forecast signals. And we also define $\mathbb{Y} \equiv \hat{\mathbb{Y}} := [1, N_Y]$.
- $\forall k \in N_K$, $W_{in}^k \in \mathbb{R}^{N_{X_k} \times N_U}$, $W_{res}^k \in \mathbb{R}^{N_{X_k} \times N_{X_k}}$ and $W_{fb}^k \in \mathbb{R}^{N_{X_k} \times N_Y}$ are matrices mapping respectively u to x^k, x^k to x^k and y (or $\hat{y}$) to x^k. These matrices are randomly constructed but supported by a well chosen topology. We present our choice of topology in Appendix (7.1.3). They stand for the part of the topology learned in an unsupervised manner.
- The output matrix is slightly different as we gather all the reservoir factors in a unique vector $s = [1 \| u \| x^1 \| \dots \| x^{N_K}] \in \mathbb{R}^{N_T \times N_S}$ where $N_S = 1 + N_u + \sum_{k=1}^{K} N_{X_k}$. Then, the output matrix is $W_{out} \in \mathbb{R}^{N_Y \times N_S}$. This matrix is the only part of the topology learned in a supervised manner.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 7.1: An Echo State Network with $N_x = K$ factors

7.1.2 Cost function

We are interested in the following temporal task: to find a model mapping u to y such that it minimises a given error measure. Mathematically, we can define a function ϕ corresponding to the network in Figure (7.1) as follows:

$$\begin{cases} \phi : T \times U \rightarrow Y \\ \phi : t, u \rightarrow \hat{y} = g(W_{out}[1 \| u(t) \| x(t)]) \\ x : T \times U \times X \times Y \rightarrow Y \\ x : t, u, x, y \rightarrow x(t) = f(W_{in}u(t) + W_{res}x(t-1) + W_{fb}y(t-1)) \end{cases} \tag{7.1.1}$$

Note, x behaves as a simple network but we will introduce a more complex function later.
We recall the definition of the Root-Mean-Square Error (RMSE) between the train and target vectors:

$$E(y_{train}, y_{target}) = \sqrt{\frac{1}{T \times N_y} \sum_{t=1}^{T} \sum_{i=1}^{N_y} \left(y_{target,i}(t) - y_{train,i}(t)\right)^2} \tag{7.1.2}$$

7.1.3 Random generation of the input and feedback matrices

These matrices are chosen randomly as suggested by the ESN framework. They can be initialised by using a particular random variable distribution, such as, the uniform or Gaussian distribution. In our experiments we choose to use a Gaussian distribution. We also consider a scaling parameter for each matrix in order to optimise them, denoted by $s_{in}, s_{bias}, s_{res}$ and s_{fb}. As discussed in Section (1.3), the feedback scaling parameter is of key importance in our network as the reservoir needs to recall its last forecasting value in the best manner regarding our cost function defined in Equation (7.1.2).

7.1.4 Topology of the reservoir matrix

Using a completely random network leads the reservoir structure to be poorly understood. Rodan et al. [2012] demonstrated that simple cycle reservoir (SCR) shows comparable performance to the randomised one. Further, they

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

have introduced a new model adding regular jumps to the SCR, called cycle reservoir with jumps (CRJ). The latter has shown superior performance compared to the traditional ones in non-linear system identification, more particularly in time series prediction. We consider this last model in our system. For example, if each of our reservoir factor was made of six neurons, the reservoir would be as in Figure (7.2).

Figure 7.2: CRJ model for a 6 neurons reservoir factor

In the CRJ model, four parameters are considered for each factor: the cycle weight r_c, the jump weight r_j, the jump level telling at which neuron the jumps start and the jump sizing how many neurons are jumped. Given these parameters, the reservoir is entirely deterministic. Considering Figure (7.2) above, the reservoir matrix is constructed as follows:

$$W_{res} = \begin{bmatrix} 0 & 0 & r_j & 0 & r_j & r_c \\ r_c & 0 & 0 & 0 & 0 & 0 \\ r_j & r_c & 0 & 0 & r_j & 0 \\ 0 & 0 & r_c & 0 & 0 & 0 \\ r_j & 0 & r_j & r_c & 0 & 0 \\ 0 & 0 & 0 & 0 & r_c & 0 \end{bmatrix}$$

7.1.5 Reservoir dynamics

In a temporal task, it is important to constantly update the kernels with a function taking both new input and the previous kernels. This leads to the notion of dynamical short-term memory. The state vector x is updated at every time $t \in \mathbb{T}$ using the following recursion equations:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\forall k \in \mathbb{K} \quad \begin{cases} \tilde{x}^k(t) = \delta^k S_{a^k}(W^k_{in} \cdot [1; u(t)] + W^k_{res} \cdot x^k(t-1) + W^k_{fb} \cdot y(t-1) + \epsilon^k) \\ \qquad\qquad + (1-\delta^k)(W^k_{in} \cdot [1; u(t)] + W^k_{res} \cdot x^k(t-1) + W^k_{fb} \cdot y(t-1) + \epsilon^k) \\ x^k(t) = (1-\gamma^k)x^k(t-1) + \gamma^k \tilde{x}^k(t) \end{cases} \tag{7.1.3}$$

where γ^k is the learning rate of the k-th kernel telling how much it learns from what has happened in the previous state as new data come in. ϵ^k is a matrix noising the kth kernel and thus preventing overfitting. It is monitored by a parameter determining the standard deviation of the Gaussian distribution. a^k is the active rate which is linked to the parametrised sigmoid function S_{a^k}. This function is applied element wise to our multidimensional factors. A plot is given in Figure (7.3), illustrating its behaviour as a varies.

$$S_a : x \in \mathbb{R} \to \frac{2}{1 + e^{-\frac{x}{a}}} - 1 \tag{7.1.4}$$

Figure 7.3: parametric sigmoid function

Sigmoidal functions are often used in machine learning to squash values to their limits. They are widely used in classification tasks. The reason being that this function is a way of assigning the negative value -1 or positive value $+1$ to a neuron in a smooth manner. The active rate a defines how important this squashing is. Further, δ^k monitors the importance of this squashing given the neuron's value.

For a given time t, the resulting $x(t)$ is called echoes of its input history $\{u(1) \dots u(t)\}$. We can set $\gamma = \alpha \times \Delta t$, where Δt is the time interval at which the data is coming into the system and α is a parameter. The smaller α and Δt are, the more the reservoir reacts slowly to new input. Note that from a signal processing point of view, the exponential moving average in Equations (7.1.3) acts as a low-pass filtering of its activations with cutoff frequency:

$$f_c = \frac{\alpha}{2\pi(1-\alpha)\Delta t}$$

7.2 Optimisation methods

We sum up the parameter set in the following picture and table. For each parameter, we provide its location, its characteristics such as its type (micro or macro) and its RMSE variability impact.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 7.4: Parameters to be optimized.

	Type		RMSE Influence		
Parameter	Micro	Macro	Low	Med	High
r_c	×		×		
r_j	×		×		
$jump$	×		×		
$jumpLevel$	×		×		
$wavelet_{th}$		×			×
β	×		×		
N		×	×		
s_{in}	×			×	
s_{bias}	×		×		
s_{res}	×			×	
s_{fb}	×			×	
γ	×				×
δ	×		×		
a	×				×
ϵ	×		×		

Table 7.1: Parameter characteristics. Micro/Macro corresponds to a matrix ladder (a parameter is micro if it defines only its elements, macro if it defines its shape). Thanks to a graph of the RMSE through the optimisation process, we can classify the influence of each parameter.

Note that some parameters depend on the factor's number. The number of parameters can grow very fast as we increase this number. Only β and $wavelet_{th}$ are independent of the reservoir shape. If we define m_f as the factors' number,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

then we have the following number of parameters:

$$n_{param} = 13 * m_f + 2 \tag{7.2.5}$$

We have focused on a 2-factor reservoir in our study as it provides two high dimensional signals and at the same time keeps the optimisation time relatively low. Therefore, the number of parameters is 28. This is relatively high and we cannot rely on an exhaustive line search, as it would take too much time to run, and also the system would be over-optimised, leading to overfitting the data.

7.2.1 Random Generation: GEN

One way of solving the above problem is to consider the following functions:

- global optimisation: it generates uniformly random values inside a space $globalMax$ times in a same loop and update the best parameters upon the RMSE obtained on the validation sample by running training and validation.
- local optimisation: exactly the same as the global one with $localMax$ times, except that it is now restricted to a smaller space (20% of the global one).
- cross optimisation: it consists in optimizing the parameters one by one $crossMax$ times and see if there is any improvement on the validation sample. It also performs two local optimisations at the end (one at 20% followed by one at 10%).
- While loop: it combines a more exhaustive global optimization ($whileMax$ times) with local and cross optimisations if there is improvement in the global.

***Remark* 7.2.1** *Note that at each call of any of these functions, new training and validation running are performed.*

Algorithm 12 Optimisation Strategy 1: GEN

Require: get $whileMax$.
Require: set $count \leftarrow 0$, $best_{RMSE} \leftarrow \infty$.

```
RMSE ← global()
RMSE ← local()
RMSE ← cross()
while count < whileMax do
    p ← random()
    node ← Reservoir(p)
    prenode ← LeakyIntegrator(node)
    prenode.train()
    RMSE ← prenode.valid()
    if RMSE < best_RMSE then
        RMSE ← cross()
        RMSE ← local()
    end if
    count ← count + 1
end while
return RMSE
```


This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 7.5: Random optimisation strategy with 2 parameters. The global optimisation finds an optimal point (purple). Then a local optimisation is done in the blue area where another optimum is found (blue). Then comes the cross optimisation where an optimum is firstly found along x_2 (orange) and secondly along x_1 (red). The $while$ loop part is not represented here in order to lighten the figure but the reader can easily see what it consists in.

***Remark* 7.2.2** *In our experiments, we note that the while loop is often useless when the cross optimisation is long enough.*

7.2.2 Stochastic Gradient Descent: SGD

The idea of this optimisation strategy is to still perform a global search and also try to find as quickly as possible the local optimum by performing a stochastic gradient descent.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 7.6: SGD optimisation strategy with 2 parameters. For every global generation (purple) we refine it by changing only a few parameters in order to get to a local optimum (red).

Tang et al. [2014] implemented this strategy by changing only the parameters r_c and r_j. They claimed that this method converges faster to an optimum and does not affect the accuracy. In this section, we lay out this method for the following set of parameters: r_c, r_j, δ, γ and a. The authors also define the reservoir matrix W_{res} with a cycle reservoir with jumps (CRJ). The idea of their method is to train the system with a hybrid optimisation strategy:

- Readouts: the output weights of the matrix W_{out} is trained using Equation (6.3.11).
- Reservoir: each parameter θ is optimised on a grid search. E is defined as in Equation (7.1.2).

$$\frac{\partial E}{\partial \theta} = 2 \sum_{t=t_0+1}^{t_1} (\hat{y}(t) - y(t)) W_{out} \frac{\partial x(t)}{\partial \theta} + \underbrace{2 \sum_{t=t_0+1}^{t_1} (\hat{y}(t) - y(t)) x(t) \frac{\partial W_{out}}{\partial \theta}}_{=0} \tag{7.2.6}$$

where θ is the parameter we optimise. Indeed, the second term is null as we use W_{out} from Equation (6.3.11), which implies that $\frac{\partial E}{\partial \theta} = \sum_{t=t_0+1}^{t_1} (\hat{y}(t) - y(t)) x(t) = 0$.

***Remark* 7.2.3** *Note that this fact is only true when W_{out} is constant, which is not the case in online algorithms. Thus, we will mainly use the GEN optimiser in our study in order to fairly compare the different methods.*

Recall the two equations updating a factor of the reservoir at time t:

$$\begin{cases} \tilde{x}(t) = \delta S_a(W_{in} \cdot [1; u(t)] + W_{res} \cdot x(t-1) + W_{fb} \cdot y(t-1) + \epsilon) \\ \qquad\quad + (1-\delta)(W_{in} \cdot [1; u(t)] + W_{res} \cdot x(t-1) + W_{fb} \cdot y(t-1) + \epsilon) \\ x(t) = (1-\gamma) x(t-1) + \gamma \tilde{x}(t) \end{cases} \tag{7.2.7}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

From these equations we can derive $\frac{\partial x(t)}{\partial \theta}$:

$$\begin{cases} \frac{\partial x(t)}{\partial r_c} = (1-\gamma)\frac{\partial x(t-1)}{\partial r_c} + \gamma(1-\delta+\frac{\delta}{a}S_a(\omega)(1-S_a(\omega)))\odot(\frac{\partial W}{\partial r_c}x(t-1)+W\frac{\partial x(t-1)}{\partial r_c}) \\ \frac{\partial x(t)}{\partial r_j} = (1-\gamma)\frac{\partial x(t-1)}{\partial r_j} + \gamma(1-\delta+\frac{\delta}{a}S_a(\omega)(1-S_a(\omega)))\odot(\frac{\partial W}{\partial r_j}x(t-1)+W\frac{\partial x(t-1)}{\partial r_j}) \\ \frac{\partial x(t)}{\partial \delta} = (1-\gamma)\frac{\partial x(t-1)}{\partial \delta} + \gamma(S_a(\omega)-\omega) \\ \frac{\partial x(t)}{\partial \gamma} = -x(t-1)+\tilde{x}(t) \\ \frac{\partial x(t)}{\partial a} = (1-\gamma)\frac{\partial x(t-1)}{\partial a} - \gamma\delta\frac{u}{a^2}S_a(\omega)(1-S_a(\omega)) \end{cases} \tag{7.2.8}$$

where $\omega = W_{in}\cdot[1;u(t)] + W_{res}\cdot x(t-1) + W_{fb}\cdot y(t-1) + \epsilon$. Then we update parameter θ using the following equation:

$$\boxed{\theta_{n+1} = \theta_n - \alpha_\theta * \frac{\partial E}{\partial \theta_n}} \tag{7.2.9}$$

Algorithm 13 Optimisation Strategy 2: SGD

Require: get $whileMax$, SGD_{Max}, V (parameters' set).
Require: set $RMSE$, count $\leftarrow 0$, $whileCount \leftarrow 0$, $SGD_{count} \leftarrow 0$ **p**.

```
while whileCount < whileMax do
    p ← random()
    while SGD_count < SGD_Max do
        for θ in V do
            θ = θ − α_θ * ∂E/∂θ
        end for
        node ← Reservoir(p)
        prenode ← LeakyIntegrator(node)
        prenode.train()
        RMSE ← prenode.valid()
        SGD_count ← SGD_count + 1
    end while
end while
return RMSE
```

7.3 Adaptive filters for online learning

There exists several ways of defining an online training algorithm, such as gradient descent and recursive least squares (see Haykin [2000], Bishop [2006]).

7.3.1 A review of adaptive filtering theory

We consider the adaptive filtering theory and justify its ability to improve the offline model presented in the previous part. Douglas [1999] gave the following definition:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 7.3.1 *Adaptive Filter*
An adaptive filter is a computational device that attempts to model the relationship between signals in real time in an iterative manner.

Later, Steil [2004] argued that an adaptive filter is able to adapt extremely fast to the main characteristics of the task in a sort of one-shot online-learning.
In general, an adaptive filter can be identified by four different features:

1. The signal being processed by the filter. For instance, it can correspond to the daily closed prices.
2. The structure/scheme/process that is used to compute the output signal from the input signal it processes. This is represented by Figure (7.1).
3. The parameters defined in its structure/scheme/process that can be modified in an iterative manner. The aim being to modify the input-output relationship to reduce a cost function.
4. The algorithm describing how these parameters are modified through time.

Given $\mathbb{T} = [1, N_T]$ a discrete time set, for each $t \in \mathbb{T}$, the input signal $u(t)$ is fed into the reservoir, now called adaptive filter, which computes an output signal $\hat{y}(t)$. The reservoir defines parameters that can adjust the values of $\hat{y}(t)$. At time $t+1$ we know the real world value of the output, therefore it is natural to define the difference signal as follows:

$$\forall t \in [1, T]\,,\ e(t) = y(t) - \hat{y}(t)$$

This signal, described in the diagram in Figure (7.7), is called the error signal. As time t is incremented, we hope that our adaptive filter learns from the environment and gets better and better at matching the response signal $y(t)$ through an adaptation process.

Figure 7.7: Scheme of the adaptive filtering purpose

There exists several types of application such as system identification, inverse modelling and linear prediction. For our task, we will use the latter and describe it in the diagram in Figure (7.8).

Figure 7.8: Linear Predictor problem

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In our task, we have the following:

$$\forall t \in [1, T]\,,\ u(t) = y(t - \Delta)$$

with $\Delta = 1$ when the non-filtered data are fed in the adaptive filter.
The diagram above indicates that if a certain output signal $y(t)$ is desired, we can consider it as input to compute the output of our model $\hat{y}(t)$. If the system we wish to predict is stable enough, then the next forecast should be even closer.
There are several measures to quantify the task performance such as the root mean square error (RMSE). An adaptive algorithm aims at adjusting a set of parameters of the corresponding adaptive filter in order to reduce as much as possible the RMSE. Given Definition (7.3.1), an adaptive signal must define some parameters which can be adapted at every point in time to achieve this goal. From RC theory, since the significant matrix in the learning process is W_{out}, we should only adapt the latter. A general equation to model an adaptive filtering algorithm in our task is as follow:

$$W_{out}(t+1) = W_{out}(t) + \mu(t) G\left(e(t), u(t), s(t)\right)$$

where $G(\cdot)$ is a vector-valued non-linear function, $\mu(t)$ a parameter controlling the step size at time t. $e(t)$ and $u(t)$ are respectively the error and input signals evaluated at time t. Douglas [1999] also suggested *"$s(t)$ [as] a vector of states that store pertinent information about the characteristics of the input and error signals and/or the coefficients at previous time instants."* s stands as the reservoir states in our task. The author pointed out that the success or failure of an adaptive filtering algorithm heavily relies on the choice of the step size μ. Therefore, μ requires significant attention when optimising in the validation set.

7.3.1.1 Simple gradient descent

We recall the gradient descent update equation below:

$$\boxed{\omega_{i,j}(t+1) = \omega_{i,j}(t) - \mu_t \cdot \frac{\partial SE(t)}{\partial \omega_{i,j}(t)}} \tag{7.3.10}$$

where $\omega_{i,j}(t)$ is the (i,j)th element of the output matrix W_{out} at time t. We also recall the equation computing the forecast at time t:

$$\hat{y}(t) = g\left(W_{out} s(t)\right) \tag{7.3.11}$$

where $g \in C^1(\mathbb{R})$ and applies to a vector in an element-wise manner and $\forall t \in \mathbb{T}\ \ s(t) = [1 \| u(t) \| x(t)]$.
We derive $\frac{\partial E(t)}{\partial \omega_{i,j}(t)}$ below using the framework defined in Appendix (5.1.4).

$$SE(t) = \sum_{i=1}^{N} \left(y_i(t) - \hat{y}_i(t)\right)^2 \tag{7.3.12}$$

$$\Rightarrow \quad \frac{\partial SE(t)}{\partial \omega_{i,j}(t)} = -2 \sum_{i=1}^{N} \left(y_i(t) - \hat{y}_i(t)\right) \frac{\partial \hat{y}_i(t)}{\partial \omega_{i,j}(t)} \tag{7.3.13}$$

$$\Rightarrow \quad \frac{\partial SE(t)}{\partial \omega_{i,j}(t)} = -2 \sum_{i=1}^{N} \left(y_i(t) - \hat{y}_i(t)\right) g'\left(W_{out} s(t)\right) s_j(t) \tag{7.3.14}$$

7.3.1.2 Recursive least squares

The Recursive Least Squares (RLS) algorithm can be used for iterative computation of the inverse of the correlation matrix R_{ij}^{-1} and the readout weights ω (see Appendix (6.3.1)). It can be summarised as follows:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$
\begin{aligned}
\kappa(t) &= \frac{\nu^{-1}\Gamma(t-1)x(t)}{1+\nu^{-1}x^{\top}(t)\Gamma(t-1)x(t)} \\
\omega(t) &= \omega(t-1)+\kappa(t)\left(y(t)-\hat{y}(t)\right) \\
\Gamma(t) &= \nu^{-1}\Gamma(t-1)-\nu^{-1}\kappa(t)x^{\top}(t)\Gamma(t-1)
\end{aligned}
$$

where $\kappa(t)$ is the gain vector and $\Gamma(t)$ is the estimate of the inverse of the correlation matrix. The forgetting factor ν is set to 1. This method ensures faster convergence of the readout weights but is much more computationally intensive than the simple gradient descent algorithm.

7.3.2 Online algorithm

We are now going to describe the online algorithm for real time learning of output matrix W_{out}. While we have considered the actual output of the financial time series y, it is more relevant to take the filtered signal as the target signal. Indeed, this signal is by definition smoother and therefore more continuous in time.

Algorithm 14 RC online Training

Require: choose a gradient descent algorithm $\mathbb{A}$

```
W_out ← Algorithm ( 11)
set SE ← 0
for t ← 0 to T_valid - 1 do
    update each reservoir factor using Equation ( 7.1.3).
    compute forecast using Equation ( 7.3.11).
    SE ← SE + (y(t + 1) − ŷ(t + 1))^2
    update W_out with Equation ( 7.3.10) using A.
end for
```

In Algorithm (14) we use $\mathbb{A}$ as the scaled conjugate gradient implemented in GNU Scientific Library [2017]. This method is very fast for a well-chosen initial guess and it makes sense to suppose that the output matrix should be close to its previous state as the signal we try to calibrate is smooth.

***Remark* 7.3.1** *We can also replace y in Equation (7.1.3) with its last feedback $\hat{y}$. Consequently, the reservoir has more information about how accurate its last forecast was.*

7.4 Some results

In order to gain some insight on the capability of neural networks to reproduce some dynamics, we will test the Reservoir Computing (RC), in particular the implementation given in Section (7.1), on the Mackey-Glass signal and some FX time series. The model will be tested both with offline and online algorithms.
We reproduce below some results on the Mackey-Glass signal and FX series obtained by a student of ours who did his MSc thesis at QFL (see Bouizi [2017]).

7.4.1 Using Offline algorithms

7.4.1.1 The Mackey-Glass series

In the academic literature, it is found that pattern generation can be easily implemented and produces very good results. However, the Mackey-Glass series prediction task is more complex and requires a large reservoir and more elaborate

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

training algorithm (see Wyffels et al. [2009], Antonik et al. [2016]).

Figure 7.9: Mackey-Glass time series forecasting. On the left hand side from top to bottom: Training and Validation samples. On the right hand side from top to bottom: Testing and parameters optimisation through the GEN optimisation process. A black dotted vertical line indicates the beginning of a global or $while$ loop optimisation. A gray one corresponds to a local one and a purple one indicates a cross optimisation.

Method	RMSE	Ranking
Neural tree (see Chen et al. [2004])	0.0069	5
PNN (see Tan [1995])	0.0059	4
NN + Wavelet Denoising (see Tan [2009])	0.0028	3
NN + Wavelet Packet Denoising (see Tan [2009])	0.0024	2
ESN offline	<u>0.00065</u>	1

Table 7.2: Comparison of our results with other research papers on the testing sample.

From Table (7.2) above, we can conclude that our method is very accurate at forecasting the Mackey-Glass time series and clearly outperforms classical methods. It is important to note that GEN optimisation method (see Appendix (7.1.3)) has reduced the RMSE from 0.00545518 to 0.000474968 on the validation sample. One of the reason for getting better results is that we optimise a large number of parameters in a proper manner without overfitting the data and in a relatively short period of time (approximately 00:01:36).

***Remark* 7.4.1** *It is not surprising to get good accuracy as this signal exhibits clear information without any noise. In this experiment, we only assess the ability of our neural network to incorporate its behaviour and to reproduce it step by step as redundant information comes in.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

7.4.1.2 Financial time series

Following the same approach on financial time series, we reproduce some results on the FX markets. We perform a GEN optimisation (see Section (7.1.3)) with offline training using the following parameters: $(globalMax, crossMax, localMax, whileMax) = (20, 20, 20, 0)$. Considering Remark (7.2.2), we choose the following setting $whileMax = 0$. We study the specific FX pair: EURGBP from 2010-01-01 to 2017-05-01. From Figure (7.10) below, we observe good results on the training and validation samples with a relatively low RMSE and a good score (almost 60%). However, Figure (7.10) suggests that testing is not satisfying, with a high RMSE at 0.0440 and a score below 50%.

Figure 7.10: EURGBP time series forecasting. On the left hand side from top to bottom: Training and Validation samples. The filter (green curve) is used as input on these samples. On the right hand side from top to bottom: Testing and parameters optimisation through the GEN optimisation process. It is important to precise that the filter (green curve) is not used during testing and therefore is purely informative.

So far, we have worked with a single FX pair. In order to perform a broader study, we consider a pool of instruments from the same market and focus on two cases: one with filtered data and one with non-filtered data. We have gathered our results and computed the mean of the RMSE and Score for each sample.

	Filtered Input			Non-Filtered Input		
	Training	Validation	Testing	Training	Validation	Testing
RMSE	0.0199	0.0282	0.0290	0.0241	0.0336	0.0237
Score	0.6207	0.6692	0.5045	0.4860	0.5196	0.5126

Table 7.3: Comparison between filtered input and non-filtered input results on 71 FX pairs.

From Table (7.3), one can clearly observe that we get better learning when using filtered input data. However, we can not use a denoising technique when testing the data as it would result in data-snooping bias. Thus, we are left with using market data, which results in poor performance (slightly above 50%). Intuitively, we have trained a reservoir with filtered data. As long as the new input data is relatively smooth, the reservoir can produce good predictions.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

However, in presence of noise the reservoir state is too much disrupted and gets too far away from a stable state. When using non-filtered data, the table shows that there is no learning. It corresponds to case (3) above where the network is trained with an attractor-based model but exploitation is ran on a transient process.

7.4.2 Using online algorithms

In the case of the Mackey-Glass time series discussed in Section (7.4.1), we have shown that the offline algorithm was very good at calibrating and predicting this signal. However, we have seen that the performance was downgraded when adding a noisy component to it (equivalent to financial time series).
Both academics and practitioners proposed to smooth financial data with filtering methods. For instance, wavelets were considered as they can represent any signal in a smooth manner and, as such, provide a certain stability (see Appendix (23.3)). Further, it was suggested that they give information about the past and future at any time t in the training and validation samples, teaching the model how to behave. A good overview of the application of wavelets in economics and finance is given by Ramsey [1999].

7.4.2.1 The Mackey-Glass series

For simplicity of exposition, we consider the following framework:

$$\forall t \in \mathbb{T} \qquad f(t) = f^*(t) + \epsilon(t) \tag{7.4.15}$$

where ϵ is a Gaussian white noise with an appropriate constant width. We also suppose that there exists a wavelet $\hat{f}^*$ capable of approximating f^*. We want to show that learning with an online algorithm produces better approximation results than with an offline algorithm. We consider the Mackey-Glass signal presented in Section (1.6.1.1). In the training and validation samples, f^* is used as target y in the Algorithm (14). Again, we cannot use it in the testing sample as it would result in data-snooping bias. Therefore, the green signal is only informative. The signal which is used in the testing sample by Algorithm (14) is the one coloured in magenta in Figure (7.13). It is the result of Algorithm (68) applied at every time t in the testing sample (from the beginning of validation).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 7.11: Noised Mackey-Glass online forecasting on the training sample. Top Left: Original, target filtered signals and forecasts. We have also plotted the local accuracy forecast (+1 if real slope is well forecast, -1 if not). Bottom Left: Reservoir states L2-norm from offline and online algorithms. Top Right: Normalised output weights evolution $\frac{||W_{out}(t)-W_{out}(t-1)||}{W_{out}(0)}$. where $W_{out}(0)$ is the offline matrix calibrated on the training sample. Bottom Right: error $cumulRMSE_{on} - cumulRMSE_{off}$.

From Figure (7.11) above, one can see that the denoised signal using Algorithm (68) is very smooth and has the same shape as the original Mack-Glass time series. Further, the learning Algorithm (14) leads to very good prediction of the green signal. This good learning is guaranteed by the combination of smooth states (plot on the bottom left, better visualisation on Figure (7.13)) and the adaptive filtering technique refining the output weights at each point in time (plot on the top right). This technique results in a very good approximation of the filtered signal and clearly outperforms the offline algorithm. Overall, we get a very good RMSE at 0.048246 and score at 73.88%. Note that we have used only a few iteration in this simulation: $(globalMax, crossMax, localMax) = (1, 1, 1)$. It indicates that a lot of reservoirs can be appropriate for our task, adding more flexibility. Also it is very computationally efficient as a good learning can be produced in less than 11 sec.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 7.12: Noised Mackey-Glass online forecasting on the validation sample. See Figure (7.11) for an advanced description.

From Figure (7.12) above, we see that the good prediction performance is preserved on the validation sample and that the algorithm produces roughly the same results than the one obtained on the training sample.

Figure 7.13: Noised Mackey-Glass online forecasting on the testing sample. See Figure (7.11) for an advanced description. We have added the filtered used by the algorithm in magenta. The green one is informative only.

Figure (7.13) shows that the magenta signal produces good results as the states remain stable. However, the results of the online algorithm against the offline one are not as good. Further, the output weights are still changing (from 5% to 25%), indicating that the online algorithm cannot forecast constant parameters in time. This is undesirable as we would like our reservoir to have good predictive properties when only feeding it with the new filtered input.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

7.4.2.2 Financial time series

We now assess the performance of the online algorithm on market data. Again, we use our pool of 71 FX pairs from 2010-01-01 to 2017-05-01.

Figure 7.14: EURUSD online forecasting on the training sample.

Figure (7.14) above demonstrates good learning and good performance of the online algorithm compared to the offline one. However the output matrix is sometimes unstable. For instance, a peak of nearly 6% has occurred between 2012-06 and 2012-12, which is well above the average amplitude of the relative output matrix variation. It often happens during regime shift. Further, when we look at EURGBP on the 23rd of June, 2016 (Brexit), we see that the adaptive filtering has produced a peak of 4%. Overall, there is good learning with a RMSE at 0.011733 and a Score at 63.21%. The algorithm ran in approximately one minute.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 7.15: EURUSD online forecasting on the validation sample.

From Figure (7.15) above, we note that good performance is still maintained on the validation sample.

Figure 7.16: EURUSD online forecasting on the testing sample.

On the testing sample, the performance is degraded as the filter we used has no-longer information about the future. Therefore, it does not move like the green one. Nonetheless, the states norm is still smoother than the one obtained from the offline algorithm, which is desired when learning time series. Further, we observe that the output weights are varying a lot, although with small amplitude (most of the time below 0.5%). Thus, we conclude that the forecast has not reach the nearest stable point (which would correspond to the next value of the filtered series we use).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

	Training	Validation	Testing
RMSE	0.0213	0.0314	0.0813
Score	0.6078	0.6694	0.4910

Table 7.4: Global assessment of the online algorithm on the pool of 71 FX pairs from 2015-01-01 to 2017-05-01.

In Table (7.4), we report the results on the same FX pool as the one used previously. Our results demonstrate good learning on the training and validation samples with low RMSEs and high scores. However, some issues remain on the testing sample. One could argue that remaining as close as possible to the filtered series (magenta) in the testing is crucial, while not interfering with the calibrated model by modifying some matrix (particularly the output weights).

7.5 Associative Reservoir Computing

7.5.1 Paradigm

7.5.1.1 Definitions

So far, we have taught the reservoir the input to output mapping. Equation (7.5.16) introduce the function ϕ defining this mapping and carrying the forward relation between the input sample U and the output sample Y without studying its properties. We are now going to understand the function ϕ in view of making better prediction.

Definition 7.5.1 *Ambiguous relation*
Let $\phi : U \to Y$. We say that ϕ is ambiguous if it is not injective i.e. $\exists (u_1, u_2) \in U^2 | u_1 \neq u_2 \Rightarrow \phi(u_1) = \phi(u_2)$

Definition 7.5.2 *State space*
A state space is the set of all possible states [the reservoir states x in our case] of a dynamical system. Each state of the system corresponds to a unique point in the state space (see Terman et al. [2008]).

Definition 7.5.3 *Attracting set*
Let S be a dynamical system and X its associated state space. We say of a set A to be an attracting set of S if it is a closed subset of X such that $\forall a \in A, \exists x(0) = x_0 | x(t) \to a$ as $t \to +\infty$.
We call its elements attractors.

Definition 7.5.4 *Basin of attraction*
Let a be an attractor of A. We define the basin of attraction of A $B(A)$ as the set of initial conditions such that $x(t) \to a$ as $t \to +\infty$.

In associative reservoir computing, the learning process aims at shaping the reservoir dynamics in a particular attractor. A major problem when learning ϕ is that it may be an ambiguous relation. Following Reinhart [2011], we develop a method for resolving the ambiguity of ϕ using a dynamical approach.

7.5.1.2 The model

In the ARC model, we overwrite the reservoir forward representation by a joint representation involving input u and output y. To access the latter, we need a matrix $W_{fb} \neq 0$. Furthermore, we define a matrix mapping the reservoir states to the input layer and called it $W_{rec} (\in \mathbb{R}^{N_U \times N_S})$. We want to learn the mapping ψ from output to input. Figure (7.17) defines the different connections involved in this model. We have the following system:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{cases} \phi : T \times U \times X \to Y \\ \phi : t, u, x_{fwd} \to \hat{y} = g(W_{out}[1 \| u(t) \| x_{fwd}(t)]) \\ \psi : T \times Y \times X \to U \\ \psi : t, y, x_{bwd} \to \hat{u} = g(W_{rec}[1 \| y(t) \| x_{bwd}(t)]) \\ x_{fwd} : T \times U \times X \times Y \to X \\ x_{fwd} : t, u, x, y \to x_{fwd}(t) = f(W_{in}u(t) + W_{res}x(t-1) + W_{fb}\hat{y}(t-1)) \\ x_{bwd} : T \times U \times X \times Y \to X \\ x_{bwd} : t, u, x, y \to x_{bwd}(t) = f(W_{in}\hat{u}(t-1) + W_{res}x(t-1) + W_{fb}y(t)) \end{cases} \tag{7.5.16}$$

Equations (7.5.16) above define the associative system. The system is now composed of two functions ϕ and ψ mapping respectively input to output and output to input. We have also defined two states x_{fwd} and x_{bwd} corresponding respectively to the states obtained with forward model ϕ and backward model ψ.

***Remark* 7.5.1** *In practice the backward model cannot be use in our task (at least on the testing sample) as we obviously do not know the asset price we want to predict. However this framework demonstrates that we can compute its corresponding input $\hat{u}$ by simply using the last prediction $\hat{y}(t-1)$.*

Figure 7.17: ARC model.

Typically, it is often recommended to use the output at time t when learning on the training sample, we call this process teacher forcing. On the validation set, we can keep training the W_{out} and W_{rec} matrices in order to teach the reservoir how to stay close to the target signal. That is to say, we use equation 7.3.10 to update W_{out} and W_{rec}, the idea being to stay close to an attractor as time goes on. In the next subsection we lay out our algorithm for converging to an attractor at every time.

7.5.1.3 Attractor-based computation

In the Figure (7.18) below, the system sits at time t and iterates the reservoir dynamics for a certain number of time, k_{max}, using always the same input $u(t)$. This artificial dynamics causes the reservoir $x(t)$ to evolve toward a stable state. A good forecast of the output attractor can be performed using Equations (7.5.16) afterwards.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 7.18: Dynamical system model to fit an attractor.

As explained by Reinhart [2011], the dynamical approach to model ambiguous inverse problems crucially depends on the ability to shape multi-stable attractor dynamics that reflect the multiplicity of solutions.

***Remark* 7.5.2** *The attractors we try to fit are the different filters we may use to denoise the initial signal. For instance, it can correspond to several wavelets, decomposition and thresholding levels.*

Algorithm 15 Attractor Convergence Algorithm

Require: get an input data at time $u(t)$.
Require: set k_{max} as a maximum number of iteration.
Require: set ϵ as a tolerance level.

```
while k < k_max and norm < ε do
    compute x_fwd(t) using Equation ( 7.5.16).
    compute ŷ(t) and û(t) using Equation ( 7.5.16).
    norm ← ||x_fwd(k + 1) − x_fwd(k)||^2
    k ← k + 1
end while
```

In Algorithm (15) we eliminate the transient effect by iterating the states using the same input. It is as if we would have frozen the series at time t and went on feeding the reservoir, leading to convergence to an attractor. Basically, we want to fit the reservoir dynamics to this attractor in order to teach the reservoir how to reproduce its behaviour. If it is the case, then the network should not need this trick any more and it would predict well this attractor. Ambiguity can be resolved by applying Algorithm (15), causing the reservoir to settle in a particular attractor.

7.5.2 States stabilisation techniques

A major issue of our task is that only noised data are observable in the market. However, the reservoir states need to be smooth (in the sense of its norm) in order to hope learning. Indeed, if the states are stables as time goes, then the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

predicted values should be stables as well. From Table (7.3) we have seen that there is a good learning when dealing with filtered input. However we cannot use the wavelet filter in the testing sample. To offset this problem, we could imagine teaching the reservoir how to remain consistently stable while feeding it with noisy input.

7.5.2.1 Reservoir dynamics calibration to observable stable states

Suppose that we can observe stable states x^* on the training sample. We wish to find a dynamics $W_{opt} = [W_{in}\|W_{res}\|W_{fb}]$ such that it minimises the following cost function:

$$E = \sum_{t=1}^{T_{training}} ||x^*(t) - x(t)||^2 \tag{7.5.17}$$

Given u, x^* and y on the training sample, the problem is stated as follows:

$$\min_{W_{opt}} \sum_{t=1}^{T_{training}-1} ||x^*(t+1) - f(W_{opt}[u(t)\|x^*(t)\|y(t)])||^2 \tag{7.5.18}$$

Let us define

$$S^* = [[u(0)\|x^*(0)\|y(0)]\| \dots \|[u(T_{training}-1)\|x^*(T_{training}-1)\|y(T_{training}-1)]]$$

and

$$A^* = [f^{-1}(x^*(1))\| \dots \|f^{-1}(x^*(T_{training}))]$$

Then we can solve this problem using Ridge regression:

$$\hat{W}_{opt} = A^* \cdot S^{*T} \cdot (S^* \cdot S^{*T} + \lambda I)^{-1} \tag{7.5.19}$$

where λ is a well chosen regularization parameter. Given these calibrated matrices, the reservoir can now learn from noisy input while staying relatively stable.

***Remark* 7.5.3** *The filter is of key importance as it will define how the system can map noisy input to smooth states.*

In practice we want the reservoir to remain stable when fed with unseen noisy input. Thus, we must consider a large enough training sample to ensure the states remains stable for exploitation.

7.5.2.2 Learning with filtered inputs

Another idea would be to work with filtered inputs only and to directly teach the reservoir to predict the filtered data (coloured in green). Indeed, this filter can be seen as an indicator of the initial dataset. One could then argue that getting a good prediction of the filtered data could lead to good real time series forecasting if one can control the forecasting error.

Suppose that we observe stable states x^* on the training and validation samples. In these samples we can use all the information at any time, so that we can fit a filtering technique such as wavelets and feed the reservoir using the filtered inputs only. The states should then be relatively stable on the training and validation, as previously discussed. Then, using Equation (7.3.10) (for both W_{out} and W_{res}), we teach the network to fit this filtered dynamics. At the end of the validation process, we hope that our network will be well enough trained and that we no-longer need Equation (7.3.10) to obtain good forecasts. That is, we should have the functions ϕ and ψ relatively constant with respect to the parameters $(W_{in}, W_{res}, W_{fb}, W_{out}, W_{rec})$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We present in Figure (7.19) our risk management scheme when making a forecast. The idea being to use, under particular assumptions, the information at time t with the backward mapping function ψ, in order to have a control over the forecast error.

Figure 7.19: Risk management scheme.

Suppose that $\exists t_0 \in [0, T_{valid}]$ such that $\forall t \in \mathbb{T}_0 := [t_0, T_{valid}]$, $\phi(u(t)) = \hat{y}(t)$ and $\psi(\hat{y}(t)) = u(t)$. Then $\psi^{-1}_{|\mathbb{T}_0} = \phi_{|\mathbb{T}_0}$ (injective property is verified at time t if Algorithm (15) converges). Suppose also that $\phi_{|\mathbb{T}_0}$ is time-independent and that $\forall t \in \mathbb{T}_0, |u^*(t) - \hat{u}(t)| \leqslant K$. Then, we could set up the following strategy:

Algorithm 16 Out-of-sample forecasting risk management

Require: set $l_V = \sup\{|u^*(t) - \hat{u}(t)|, \forall t \in \mathbb{T}_0\}$
Require: set $l_T \leftarrow 0$
1: **while** $l_T < l_V$ **do**
2: update the reservoir x up to time t using Equations (**??**) and (**??**).
3: compute forecast $\hat{y}(t)$ and associative input $\hat{u}(t)$ using Equation (7.3.11).
4: $l_T \leftarrow |u^*(t) - \hat{u}(t)|$
5: **end while**

The ARC framework is very powerful as it allows us to evaluate the input error at time t to have an idea of the forecast accuracy at time $t + 1$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 8

Introduction to unsupervised learning

We check the performance of the Recurrent Radial Basis Function by fitting and predicting chaotic time series. This could be verified by assessing its performance on the classic Mackey-Glass series (see Figure (6.15)). We also verify performance of the models on denoised financial time series. We let the Linear Regression (LR) model be the benchmark (see Section (12.1.1)). We reproduce below some results on the Mackey-Glass signal obtained by a student of ours who did his MSc thesis at QFL (see Wang [2017]).

8.1 Batch intrinsic plasticity

8.1.1 Description

So far there are no generalised unsupervised learning algorithms that are powerful enough to be able to fully train ESN reservoirs, although several attempts were made (see Lukosevicius [2012b]). However, sometimes pre-training methods in simple machine learning models could be enlightening and useful as well in complex models such as ESN.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 8.1: Extreme Learning Machines

Extreme Learning Machine (ELM) is a simple type of feedforward neural network (see Section (5.1.3)). It has only one hidden layer and no recurrent part. Similarly to ESN, its task performance greatly depends on random initialisation of the input weight matrix $\mathbf{W}_{in}$. Without additional tuning strategies, in some cases random generations can lead to saturations of reservoir states $\mathbf{X}(t)$, where most of the entries are equal to -1 or 1, or early convergence where $\mathbf{X}(t) = \mathbf{X}(t+1)$ when t is small.

Batch Intrinsic Plasticity (BIP) (see Neumann et al. [2011]) was introduced to adapt the activation functions so that the desired distribution for $\mathbf{X}(t)$ could be realised.

Suppose ELM has the following update equations:

$$\mathbf{X}_i = f(a_i \mathbf{W}_{in}^i \mathbf{U} + b_i) \tag{8.1.1}$$

$$\hat{\mathbf{Y}} = \mathbf{W}_{out}\mathbf{X} \tag{8.1.2}$$

where $\mathbf{U} = (U(1), U(2), ..., U(nbDataPoints)) \in \mathbb{R}^{N_u \times nbDataPoints}$ is input vector, $\mathbf{W}_{in}^i$ is the ith row of input weight matrix $\mathbf{W}_{in} \in \mathbb{R}^{N_x \times N_u}$, and $\mathbf{a} \in \mathbb{R}^{1 \times N_x}$ and $\mathbf{b} \in \mathbb{R}^{1 \times N_x}$ are scalar vectors to be determined by BIP.

Consider the activation function $f = tanh$, then $X = [-1, 1]$ is an important range where $Y = tanh(X)$ is virtually linear. Outside this range, especially when $|X| > 2$, Y will be very close to -1 or 1, which can cause saturations of $\mathbf{X}(t)$ if $\mathbf{X}_i(t) = f(\mathbf{W}_{in}^i \mathbf{U}(t))$. Fortunately, with scalar vectors $\mathbf{a}$ and $\mathbf{b}$, this situation may be mitigated.

Suppose $f = tanh$, $\mathbf{S}_0 \in \mathbb{R}^{N_x \times nbDataPoints} = \mathbf{W}_{in}\mathbf{U}$ and $\mathbf{X} \in \mathbb{R}^{1 \times N_x}$ has N_x positions for elements. Then, for the ith position, the BIP will do the following:

1. Uniform samples are drawn from $[-1, 1]$ and sorted in ascending order, the result is collected by target vector $\mathbf{T} \in \mathbb{R}^{1 \times nbDataPoints}$.

2. The ith row of $\mathbf{S}_0$ is sorted in ascending order and collected by vector $\mathbf{S}$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3. Take **S** as input and **T** as output, implement a linear regression to get coefficients a_i and b_i.

Finally, iterations of i from 1 to N_x will give vectors **a** and **b**. The corresponding pseudo code can be found in the Algorithm (17) below.

Algorithm 17 BIP Training Algorithm

Require: : Import input vectors $\mathbf{u} = (u(1), u(2), ..., u(nbDataPoints))$

```
S_0 = W_in U
for i = 1 to N_x do
    S ← sorted ith row of S_0
    Construct concatenation φ = [S; (1, 1, ..., 1)] ∈ R^{2×nbDataPoints}
    Draw targets T = (T(1), T(2), ..., T(nbDataPoints)) from desired distribution
    T ← sorted T
    v_i = [a_i, b_i] = T * pinv(φ)
end for
Return v
```

Some comments are given here:

- For f=sigmoid function, the range for uniform sampling could extend to $[-2, 2]$ due to the shape of sigmoid function.
- The input and output vectors are sorted before linear regression due to the fact that the function $\mathbf{Y} = a_i\mathbf{X} + b_i$ is set to be monotonic increasing.
- Apart from uniform distributions, the above procedure also applies to other bounded distributions. For unbounded distributions such as Gaussian, things need to be changed a little bit. Drawing normally distributed samples **T** and applying $tanh$ on it clearly no longer works since the samples are not bounded between $[-1, 1]$. At this stage, one possibility to get round would be that the normal distribution is applied on $f(\mathbf{T})$, that is, the reservoir state **X**, instead of **T**. Hence, when **X** is set to be normally distributed, f^{-1} needs to be computed. The result works as new output, say, $\mathbf{T}_2$, and linear regression can then be constructed between **S** and $\mathbf{T}_2$.

Now, this algorithm could also be used on ESN as well. Consider standard ESN with leak rate $\alpha = 1$, and the following update equation:

$$\mathbf{X}(t) = f(\mathbf{W}_{in}[1; \mathbf{U}(t)] + \mathbf{W}\mathbf{X}(t-1)) \tag{8.1.3}$$

This time, suppose design matrix $\mathbf{X} = (X(1), X(2), ..., X(nbDataPoints)) \in \mathbb{R}^{N_x \times nbDataPoints}$, there are three ways of setting $\mathbf{s}_0$:

- $\mathbf{S}_0 = f^{-1}(\mathbf{X})$
- $\mathbf{S}_0 = \mathbf{X}$
- $\mathbf{S}_0 = \mathbf{W}_{in}\mathbf{U}$

Experimentally these three settings can all improve ESN. In particular, the third one will be illustrated in detail since it generally performs better than the first two settings. This is partly because the last setting has smallest root mean square error in the desired target fitting.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 8.2: desired target fitting

Now, suppose $\mathbf{S}_0 = \mathbf{W}_{in}\mathbf{U}$, other settings kept the same as ELM, the ESN reservoir state has the following update equation:

$$\mathbf{X}(t) = f(\mathbf{a}.*\mathbf{W}_{in}\mathbf{U}(t) + \mathbf{b} + \mathbf{W}\mathbf{X}(t-1))$$

Note $\mathbf{a} \in \mathbb{R}^{N_x \times 1}$ and $\mathbf{W}_{in}\mathbf{U}(t) \in \mathbb{R}^{N_x \times 1}$, so $.*$ here means element wise multiplication. For example,

$$\begin{pmatrix} a \\ b \end{pmatrix} .* \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} ac \\ bd \end{pmatrix}$$

8.1.2 Some results

Empirically, unlike the case with original ESN, normalisation barely changes the task performance of BIP-attached ESN, while desired target range matters more. Hence, experiments are done on the stock MMM using different target ranges:

	f=tanh	f=sigmoid
Uniform Range $[-2, 2]$	$90, 0.00247, 0.00148$	$170, 0.00198, 0.00130$
Uniform Range $[-1, 1]$	$120, 0.00234, 0.00147$	$150, 0.00199, 0.00109$
Uniform Range $[-0.5, 0.5]$	$130, 0.00213, 0.00145$	$440, 0.00206, 0.00122$
Uniform Range $[0, 1]$	$110, 0.00207, 0.00134$	$450, 0.00206, 0.00125$

Similarly to the previous experiment on ESN, the triplet in the table still means N_x, training error and test error respectively. Seen from the table,

- Sigmoid in general surpasses tanh on task performance.
- For f=sigmoid, that the range $[-1, 1]$ works better than $[-2, 2]$ may be due to specific setting of $\mathbf{S}_0 = \mathbf{W}_{in}\mathbf{U}$: Since $\mathbf{X}(t) = f(\mathbf{a}.*\mathbf{W}_{in}\mathbf{U}(t) + \mathbf{b} + \mathbf{W}\mathbf{X}(t-1)) = \mathbf{a}.*\mathbf{s}_0 + \mathbf{b} + \mathbf{W}\mathbf{X}(t-1))$, $\mathbf{a}.*\mathbf{S}_0 + \mathbf{b}$ only partly contributes to calculation of $\mathbf{X}(t)$, unlike in case of ELM where $\mathbf{X}(t) = f(\mathbf{a}.*\mathbf{W}_{in}\mathbf{U}(t) + \mathbf{b})$ and $\mathbf{a}.*\mathbf{S}_0 + \mathbf{b}$ fully determines $\mathbf{X}(t)$. Moreover, $X = [-1, 1]$ is a range where $f(X)$ is steepest, hence elements of $\mathbf{X}(t)$ will be more spread out there.
- The best test error 0.00109 comes from sigmoid with uniform range $[-1, 1]$, it reduces 69% of the test error made by linear regression and 20% of the smallest test error made by original ESN. The error comparisons are displayed below:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 8.3: ESN on denoised stock returns test set

Figure 8.4: ESN with BIP on denoised stock returns test set

Also, general performance of BIP-attached ESN over 100 stocks is assessed using the following common parameter setting:

Leak Rate α	1	Spectral Radius	0.65
Reservoir size N_x	150	Activation function	sigmoid
Reservoir Size $\mathbf{W}_{in}$	$[-2, 2]$	Reservoir Weight Matrix $\mathbf{W}_{res}$	$[-0.5, 0.5]$
Reservoir Density	0.5	Regularisation term β	10^{-2}

Again, the desired target range used is $[-1, 1]$, which gives the following error table:

Errors over stocks	Average Training error(RMS)	Average Test error(RMS)
Linear Regression	0.00809	0.00799
ESN	0.00498	0.00515
ESN with BIP	0.00397	0.00459

Simple calculations show that BIP-attached ESN reduces on average 20% of training error and 11% of test error made by plain ESN. This observation underpins BIP as a promising tool for improving ESN.

8.2 Radial basis function

8.2.1 Description

Radial Basis Function (RBF) were first used to solve interpolation problem, that is, fitting a curve exactly through a set of points (see Powell [1987]). They were extended to perform the more general task of approximation (see Poggio

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

et al. [1990]).
RBF are certain type of functions whose value only depend on the distance from some center $\mathbf{c}$. Any function ϕ satisfying

$$\phi(\mathbf{X}, \mathbf{c}) = \phi(||\mathbf{X} - \mathbf{c}||)$$

is called a radial basis function. The norm here is usually taken as the Euclidean norm. Poggio et al. [1990] showed that RBF could be derived from the classical regularisation problem where some unknown function $\mathbf{Y} = f(\mathbf{X})$ can be approximated from the dataset $\{(\mathbf{X}_t, \mathbf{Y}_t)\}$ and some smoothness constraints. In terms of regression analysis (see Section (12.1.1)) the function f is the conditional expectation of $\mathbf{Y}_t$ given $\mathbf{X}_t$. That is,

$$\mathbf{Y}_t = f(\mathbf{X}_t) + \epsilon_t \ , \ E[\epsilon_t|\mathbf{X}_t] = 0$$

The regularisation problem is equivalent to the minimisation of the objective function

$$L(\widehat{f}) \equiv \sum_{t=1}^{T} \Big(\|\widehat{\mathbf{Y}}_t - \widehat{f}(\mathbf{X}_t)\|^2 + \lambda \|P\widehat{f}(\mathbf{X}_t)\|^2 \Big)$$

where P is a differential operator. The second term is a penalty function decreasing the smoothness of $\widehat{f}$ and λ controls the tradeoff between smoothness and fit. In its most general form, the solution to the objective function is given by

$$f(\mathbf{X}) = \sum_{i=1}^{k} \alpha_i \phi(||\mathbf{X} - \mathbf{c}_i||) + p(\mathbf{X})$$

where α_i are scalar coefficients, $p(\cdot)$ is a polynomial and k is much less than the number of observations in the sample.
Girosi et al, [1990] showed that RBFs could approximate arbitrarily well any continuous function on a compact domain. As an important branch of neural networks, RBF networks have numerous applications, including system identification and non-linear chaotic time series forecasting.

It has rather distinct properties from ESN (e.g. only require small reservoir size). Micchelli [1986] showed that even though there is a large class of possible basis functions ϕ, the most common choices are the Gaussian function $e^{-\frac{x}{\beta^2}}$ and the multi-quadrics $\sqrt{x + \beta^2}$. In this section, we adopt the Gaussian kernel:

$$\mathbf{X}_i = \Phi(||\mathbf{U} - \mathbf{c}_i||) = \exp\Big(\frac{-||\mathbf{U} - \mathbf{c}_i||^2}{(2\beta_i^2)}\Big) \tag{8.2.4}$$

Figure 8.5: Gaussian kernel

Figure 8.6: RBF

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Essentially, each time a new input $\mathbf{U}$ is imported, it is transformed into the variable $\mathbf{X}$ using radial basis function in Equation (8.2.4). Then, the multiplication of $\mathbf{X}$ with output weight matrix $\mathbf{W}_{out}$ gives the output. This time, the output weight matrix $\mathbf{W}_{out}$ is no longer trained using a supervised ridge regression. Instead, it is tuned along with other parameters. Generally, with $\mathbf{w}$ standing for $\mathbf{W}_{out}$, the triplet $\{\mathbf{c}, \beta, \mathbf{w}\}$ needs to be tuned together using a learning algorithm called *gradient descent* (see Section (18.3.1.1)), which iteratively takes steps proportional to the negative of the current gradient of the function (see Niu et al. [2014]). The update equations are as follows:

$$w_i(n+1) = w_i(n) - \eta_1 \epsilon(n) C(n) \mathbf{x}_i(n) \tag{8.2.5}$$

$$\mathbf{c}_i(n+1) = \mathbf{c}_i(n) - \eta_2 \epsilon(n) w_i(n) C(n) \frac{\mathbf{x}_i(n)}{\beta_i(n)^2} (\mathbf{u}(n) - \mathbf{c}_i(n)) \tag{8.2.6}$$

$$\beta_i(n+1) = \beta_i(n) - \eta_3 \epsilon(n) w_i(n) C(n) \frac{\mathbf{x}_i(n)}{\beta_i(n)^3} ||\mathbf{u}(n) - \mathbf{c}_i(n)|| \tag{8.2.7}$$

Where η_i, $i = 1, 2, 3$, are learning rates which control the speed of the process. The error term $\epsilon(n)$ is the difference between target value and predicted value at time n, that is,

$$\epsilon(n) = \hat{y}(n) - y(n) = \mathbf{w}(n)\mathbf{x}(n) - y(n)$$

$C(n)$ is an additional feature called stochastic data-time effective function:

$$C(n) = \frac{1}{\tau} exp(\int_{t_0}^{n} \mu(t)dt + \int_{t_0}^{n} \sigma(t)dW_t)$$

The focus is on gradient descent algorithm itself, so for simplicity $\mu(t)$ and $\sigma(t)$ are both set to be zero. Below is the pseudo code for training RBF using gradient descent:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 18 RBF Training Algorithm

Function$modelOutput = rbfTrain\mathbf{U}, \mathbf{y}, modelInputs$ $\{modelInputs \leftarrow N_x, \eta, \tau \ldots\}$
$\mathbf{c} \in \mathbb{R}^{N_u \times N_x}, \boldsymbol{\beta} \in \mathbb{R}^{N_x \times 1}, \mathbf{W}_{out} \in \mathbb{R}^{1 \times N_x}, \mathbf{yHat} \in \mathbb{R}^{1 \times nbDataPoints}$
$k = 1, error = 10^5, maxIteration = 500, tolerance = 0.008 \leftarrow$ *Initializations*
while ($error >$ tolerance **and** $k < maxIteration$) **do**
  for $j = 1$ to $nbDataPoints$ **do**
    for $i = 1$ to N_x **do**
      $\boldsymbol{\phi}(i) = \exp\left(-\frac{||\mathbf{U}(:,j) - \mathbf{c}(:,i)||^2}{2\boldsymbol{\beta}(i)^2}\right)$
    end for
    $\epsilon = \mathbf{W}_{out}\boldsymbol{\phi} - \mathbf{y}(j)$
    $C = 1/\tau$
    for $i = 1$ to N_x **do**
      $w_i = \mathbf{W}_{out}(i)$
      $\mathbf{c}_i = \mathbf{c}(:,i)$
      $b_i = \boldsymbol{\beta}(i)$
      $\mathbf{W}_{out}(i) = \mathbf{W}_{out}(i) - \eta\epsilon C\boldsymbol{\phi}(i)$
      $\mathbf{c}(:,i) = \mathbf{c}(:,i) - \eta\epsilon C w_i \boldsymbol{\phi}(i)(\mathbf{u}(:,j) - \mathbf{c}_i)/b_i^2$ $\{\mathbf{c}(i)$ *is a vector*$\}$
      $\boldsymbol{\beta}(i) = \boldsymbol{\beta}(i) - \eta\epsilon C w_i \boldsymbol{\phi}(i)||\mathbf{u}(:,j) - \mathbf{c}_i||/b_i^3$
    end for
  end for
  for $j = 1$ to $nbDataPoints$ **do**
    for $i = 1$ to N_x **do**
      $\boldsymbol{\phi}(i) = \exp\left(-\frac{||\mathbf{U}(:,j) - \mathbf{c}(:,i)||^2}{2\boldsymbol{\beta}(i)^2}\right)$
    end for
    yHat$(j) = \mathbf{W}_{out}\boldsymbol{\phi}$
  end for
  $m \leftarrow nbPointsToIgnore + 1;$
  $\mathbf{y}_1 =$ (**yHat(:,m:end)** $- b)/a$ $\{$*the first m points are discarded*$\}$
  $\mathbf{y}_2 =$ (**y(:,m:end)** $- b)/a$ $\{$*the y values are mapped back before calculating errors*$\}$
  $error = \sqrt{\frac{1}{nbDataPoints}\sum_{n=1}^{nbDataPoints}(\mathbf{y}_1(n) - \mathbf{y}_2(n))^2}$
  $k = k + 1$
end while
Store updated $\mathbf{c}$, $\boldsymbol{\beta}$ and $\mathbf{W}_{out}$
EndFunction

Note in the above algorithm:

- Suppose $N_y = 1$, then $\mathbf{c}$ is a matrix of size $N_u \times N_x$, whenever a new input $\mathbf{U}(n)$ is imported, every column of $\mathbf{c}$ is updated using $\mathbf{c}(:,i) = \mathbf{c}(:,i) - k(\mathbf{u}(:,j) - \mathbf{c}_i)$ where k is some scalar. Similarly, $\boldsymbol{\beta}$ is a vector of size $N_x \times 1$ and $\mathbf{w}$ is a vector of size $1 \times N_x$. They will be updated accordingly.
- The algorithm will terminate if either the training error is less than the predefined tolerance, or the maximum number of iterations are hit.

8.2.2 Some results

With the following parameter settings:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Reservoir Size N_x	15	Input size N_u	1
Normalisation Range	$[0, 1]$	Learning Rate η	0.001
Data-time Effective Function C	10	Maximum Iterations	500

and these initialisations using uniform distributions on certain ranges:

$$\mathbf{c}\text{: } [0, 1] \quad \beta\text{: } [0.1, 0.3] \quad \mathbf{w}\text{: } [-0.1, 0.1]$$

Setting the tolerance to 0.008 would give the following error table and the corresponding prediction performance:

	Training error(RMS)	Testing error(RMS)
Linear Regression	0.0315	0.032
RBF	0.008	0.00771

Figure 8.7: RBF on Mackey Series test set

This first experiment costs 33 seconds, and the evolution of training errors are plotted:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 8.8: Error Evolutions

Now if the tolerance is set to be 0.007, i.e. require the training error to be no more than 0.007, then it takes 53 seconds to reach this error target:

Figure 8.9: Error Evolutions

Therefore, technically it takes 20 seconds for the algorithm to get rid of the last 0.001 training error. It means that it consumes an additional 60% of the previous running time to reduce 12% of the error. Seen from the Table, the gradient descent has fast convergence rate at the beginning. However, after 200 iterations it struggles to improve just a little bit. Now, keeping the other parameters fixed (the same random initialisations are reproducible), more experiments are

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

made by varying the learning rates:

Learning Rate η	Running Time (seconds)
0.001	53
0.002	43
0.005	63
0.0005	66

In the Table (8.2.2), only a learning rate of 0.002 speeds up the process by 10 seconds. However, further increasing the learning rate ($\eta = 0.005$) no longer helps.
In addition to learning rate, the evolvements of $\mathbf{w}$, $\boldsymbol{\beta}$ and $\mathbf{c}$ are also of interest. In the original case where tolerance=0.008 and learning rate=0.001, the final form of $\boldsymbol{\beta}$ is displayed in Figure (8.10).

Figure 8.10: bar plot of $\boldsymbol{\beta}$

Since $N_x = 15$, each bar represents each element of $\boldsymbol{\beta}$. Note that $\boldsymbol{\beta}$ is initialised to be within range $[0.1, 0.3]$. But as seen from the plot, about $\frac{1}{3}$ of the elements go far beyond its upper bound 0.3, in particular, the highest value is 1.23, which is about 4 times 0.3, and the lowest value is 0.12. Hence, if the initialisation range for $\boldsymbol{\beta}$ is expanded to $[0, 1]$ or $[0.12, 1.23]$, it is possible that the running time will be further reduced. Empirically however, neither case manages to decrease the original running time. Nevertheless, if the learning rate is adjusted to 0.002, all three cases achieve a reduction of 10 seconds on average. On the other hand, similar experimental results were obtained to the case of $\mathbf{w}$ and $\mathbf{c}$.

In general, the learning rate is a delicate parameter in machine learning. If it is set too small, it will take too much time to converge. On the other hand, if it is greater than needed, it may never converge and even diverge. See the Figure (8.11) for intuition.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 8.11: Gradient Descent-Andrew Ng

At the same time, although for gradient descent, initialisations are not required to be close-to-optimal, it has to be within a certain range for the algorithm to reach the target error within reasonable time. Ideally, the gradient descent works best on convex functions. See the Figure (8.12) for intuition. If the function has multiple peaks or several local minima, given different initialisations, gradient descent may end up finding the global minimum or being trapped in a local minimum.

Figure 8.12: Convex Functions

In summary, this section introduced a new parameter optimising method. The main advantage of gradient descent is that one can specify the desired error level beforehand so that task running time could be allocated more efficiently according to different accuracy requirements. At the same time, however, it has several limitations:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- The setting of learning rate largely depends on experience. Inappropriate setting can make it impossible to reach the target error level within reasonable time.
- Gradient descent is relatively slow close to the minimum: technically, its asymptotic rate of convergence is inferior to many other methods.
- Gradient descent fails confronting non-differentiable functions. Thus these functions may need to be bounded by a smooth function for gradient descent to work.

8.3 Recurrent RBF network

8.3.1 Standard RRBFN

Recall from Section (8.2), the radial basis function (RBF) model has the form given in Equation (8.2.4). This model is now featured with recurrent structure so that it turns into an advanced structure called Recurrent Radial Basis Function Network(RRBFN) which has the following form:

$$\begin{aligned}\mathbf{x}_i(n) &= \Phi\big(||\mathbf{U}(n) - \mathbf{c}_1^i||, ||\mathbf{X}^i(n-1) - \mathbf{c}_2^i||\big) \\ &= \exp\big(-(||\mathbf{u}(n) - \mathbf{c}_1^i||^2 + ||\mathbf{X}_i(n-1) - \mathbf{c}_2^i||^2)/(2\beta_i^2)\big)\end{aligned} \tag{8.3.8}$$

This time, the function values depend on two inputs instead of one. Two 'distances' from the two center vectors determine the output value. More specifically, its update equation is:

$$\widetilde{\mathbf{X}}_i(n) = \exp\big(-\alpha||\mathbf{W}_{in}^i - \mathbf{U}(n)||^2 - \beta||\mathbf{W}^i - \mathbf{X}(n-1)||^2\big), \quad i = 1, ..., N_x$$

where $\widetilde{\mathbf{x}}_i(n)$ is the ith element of vector $\widetilde{\mathbf{X}}(n) \in \mathbb{R}^{N_x}$; $\mathbf{W}_{in}^i$ and $\mathbf{W}^i$ are ith row of $\mathbf{W}_{in}$ and $\mathbf{W}$ respectively, α and β are scalar parameter which absorb the information of β_i in Equation (8.2.4).

8.3.1.1 Pseudo code

The following is the pseudo code for training RRBFN:

Algorithm 19 RRBFN Training Algorithm

```
Function rrbfTrain u, y, modelInputs {modelInputs ← N_x, W_in, W, X_0 etc.}
X ∈ R^{N_x × nbDataPoints} ← Design Matrix
for i = 1 to nbDataPoints do
    for j = 1 to N_x do
        X(j, i) = exp(−α||[W_in(j, :)]^T − U(:, i)||^2 − β||[W(j, :)]^⊤ − X(:, i − 1)||^2);
    end for
    z(:, i) = [1; u(:, i); X(:, i)]; {Vertical Concatenation}
end for
m ← nbPointsToIgnore + 1;
S = z(:, m : end);
D = y(:, m : end);
W_out = DS^T pinv(SS^⊤ + λI); {Ridge Regression, 'pinv' is pseudo inverse}
EndFunction
```


This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

8.3.1.2 Parameter tuning

Again, the Mackey series is used as a benchmark chaotic time series to tune the parameters of RRBFN. Compared to other chaotic time series, even noisy, it is relatively smooth and tractable. Therefore, if a machine learning model cannot succeed on training the Mackey series, then it is unlikely to have strong potential in predicting chaotic time series. Most importantly, the Mackey Series could be used to find priority of parameters. In general,

- An increase in N_u will monotonically decrease both training error and test error to a certain level. However it is not recommended to set a high value for N_u since it increases the chance of overfitting.
- W barely changes results.
- Other parameters give no obvious trends and thus need to be tuned one at a time.

It is worth mentioning that α and β have similar positions in the model, they both are scalar factors of a certain 'distance'. That is,

- α is the distance between ith row of $\mathbf{W}_{in}$ and input vector $\mathbf{u}(n)$.
- β is the distance between ith row of $\mathbf{W}$ and reservoir state $\mathbf{x}(n-1)$.

Therefore, these parameters should be treated as a pair.
First of all, assume α and β have equal importance. Start from $\alpha = \beta = 0.1$, a monotonic increase in both values until $\alpha = \beta = 0.9$ gives a general decrease in errors. In particular, $\alpha = \beta = 0.6$ is chosen to be a robust set since there is a huge drop in errors when $\alpha = \beta = 0.5$ is changed to $\alpha = \beta = 0.6$, and also this is the first time when test error is less than 10^{-3}. Nevertheless, $\alpha = \beta = 0.9$ is robust too, for it produces smallest test error when α and β are set equal.

Secondly, it remains to be seen which one has more impact on final result. Fixing other parameters, the following statistics are obtained:

α	0.6	0.6	1
β	0.6	1	0.6
Test error(RMSE)	0.000737	0.000684	0.000852

From this table, it can be observed that a 66% change in β leads to a 7% decrease in the test error, while 66% change in α results in 15.6% increase in the test error, which indicates that generally α is more influential on the error term. Then fixing $\alpha = 0.6$, careful tuning on β gives the final pair:

α	0.6
β	0.9
Test error(RMSE)	0.000669

Similarly, it is possible that some other parameters are also strongly inter-connected and should be tuned together. However, identification of this requires additional expertise and deep analysis on the behaviour of parameters.

Finally, with cautiously tuned parameters below,

[α,β]	[0.6,0.9]	Spectral Radius	1.25
Reservoir size N_x	10	Input Size N_u	1
Reservoir Size $\mathbf{W}_{in}$	$[-1, 1]$	Regularization Term	10^{-8}

Without using data normalisation, the errors are as displayed:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

	Training error(RMS)	Test error(RMS)
Linear Regression	0.0315	0.032
RRBF	0.000705	0.000648

From now on it is assumed that 0.000648 is the smallest test error that can be achieved by standard RRBFN with $N_u = 1$. This important result will be cited later for further comparison and evaluation with new methods.

8.3.2 Randomly parametrised RRBFN

Based on the conjecture that each entry of input vector $\mathbf{U}(n)$ or reservoir states $\mathbf{X}(n)$ can contribute differently to the new reservoir state $\mathbf{X}(n+1)$, modifications on distance computation are made hoping to take advantage of this hypothesis. For example, if $\mathbf{U}(n) = (U_1, U_2, U_3, U_4, U_5)^\top$ is a vector representing the log returns of the past 5 days, then U_1 could have more impact on $\mathbf{U}(n+1)$ than U_5 does. Therefore, when computing $\mathbf{W}_{in}^i - \mathbf{U}(n)$, the resultant vector $[0.1, 0, 0, 0, 0]^\top$ should be distinguished from $[0, 0, 0, 0, 0.1]$. If one favours small values of the norm $||\mathbf{W}_{in}^i - \mathbf{U}(n)||$, then the latter vector is more desirable than the first one since U_1 matters more than U_5. However, in the sense of previous Euclidean norm, the two vectors have exactly the same norm. Therefore, *weights* may be assigned to different positions within a vector to unearth and make use of the priority of importance.

However, usually the order of importance within $\mathbf{U}(n)$ is unknown and it may be constantly changing. If a specific order list is presumed and widely used over different time periods and stocks, generality of the model may be impaired. Hence, N_u being the length of vector $\mathbf{U}(n)$, a randomly assigned weight distribution is generated following the steps below:

- Draw N_u samples from standard uniform distribution $\mathrm{U}(0, 1)$.
- Calculate the sum of N_u samples, call it s;
- Divide each sample by s to get a probability distribution $\mathbf{p}$;
- Multiply each element of $\mathbf{p}$ by N_u to get final weight vector $\mathbf{w}_1$;

In the situation of equal weight distribution, all elements of $\mathbf{p}$ are the same and equal to $\frac{1}{N_u}$, and the weight vector $\mathbf{w}_1 = N_u * \mathbf{p}$ consists of all ones. This special case is equivalent to the previous standard RRBFN, where distances are measured in Euclidean norm.
The above justifications and calculations apply similarly to the norm $||\mathbf{W}^i - \mathbf{X}(n-1)||$, with N_u changed into N_x.

Fixing the optimal parameter settings for standard RRBF, iterations over different weight vector $\mathbf{w}_1$ show that the random parametrised RRBF is relatively unpredictable in improving standard RRBF. Sometimes, it could reduce the test error by 20%; On the other hand, with some extreme distribution of $\mathbf{w}_1$, this could be the other way round. Therefore, its performance largely depends on initialisation of $\mathbf{w}_1$ and is relatively unstable.

Below is a case where $\mathbf{w}_1$ is randomly initialised using rand('seed',9). In this case, randomly parametrised RRBF reduces 17% of the test error made by standard RRBF:

	Training error(RMS)	Test error(RMS)
Linear Regression	0.0315	0.032
Standard RRBF	0.000705	0.000648
Randomly Parametrized RRBF	0.000591	0.000539

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

8.3.3 Directional parametrised RRBFN

Due to the fact that the performance of randomly parametrised RRBFN greatly relies on random generation of the weight vector $\mathbf{w}_1$, one may want to alleviate, or get rid of, the random factor by generating $\mathbf{w}_1$ using non-random rules. This type of network is called directional parametrised RRBFN. It has been paid attention since in the previous experiment, some distributions of $\mathbf{w}_1$ do lead the parametrised RRBFN to a better performance than that of standard RRBFN.
In our trial, elements of $\mathbf{w}_1$ are set to be proportional to magnitudes of elements in vector $\mathbf{W}_{in}^i - \mathbf{u}(n)$. For instance, if $\mathbf{W}_{in}^i - \mathbf{U}(n) = (1,3)^\top$, then the corresponding $\mathbf{p}$ and $\mathbf{w}_1$ would be $(0.25, 0.75)^\top$, $(0.5, 1.5)^\top$ respectively. Note that only the magnitude matters. So if $\mathbf{W}_{in}^i - \mathbf{U}(n) = (-1,3)$, the corresponding $\mathbf{w}_1$ would still be $(0.25, 0.75)^\top$. Under this rule, the vector difference between $\mathbf{W}_{in}^i$ and $\mathbf{U}(n)$ is amplified. For example, if $\mathbf{W}_{in}^i - \mathbf{U}(n) = (1,3)^\top$ holds, then

- Euclidean norm would give $\sqrt{1^2 + 3^2} = \sqrt{10}$
- Under new rule, the norm is $\sqrt{0.5 * 1^2 + 1.5 * 3^2} = \sqrt{14}$

Empirically, with proper parameter settings:

- α=0.2, β=0.8
- Spectral Radius=0.9

Directional parametrised RRBF is able to improve the best standard RRBF by a meaningful amount. Refer to the following table for details:

	Training error(RMS)	Test error(RMS)
Linear Regression	0.0315	0.032
Standard RRBF	0.000705	0.000648
Randomly Parametrized RRBF	0.000591	0.000539
Directional Parametrized RRBF	0.000548	0.000479

From the table, it can be seen that the directional parametrised RRBF successfully reduces 26% of the test error made by standard RRBF. This observation is encouraging since it shows that directional parametrised RRBF may have potential to *steadily* improve RRBF with general stock data.

8.3.4 Empirical results

Firstly, the test is done on a specific stock. For consistency, stock 'MMM' is again used. After plugging in appropriate parameters, the errors are displayed below:

	Training error(RMS)	Test error(RMS)
Linear Regression	0.00472	0.0036
Standard RRBF	0.00290	0.00173
Directional Parametrized RRBF	0.00269	0.00153

Again, the above errors in the table are smallest errors (especially test error) attainable using corresponding methods. This time, the directional parametrised RRBF decreases 12% of the test error made by standard RRBF, which is a good start.

Now, with a common parameter setting, data are collected across 100 stocks:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

	Average Training error(RMS)	Average Test error(RMS)
Linear Regression	0.00809	0.00799
Standard RRBF	0.00531	0.00598
Directional Parametrized RRBF	0.00519	0.00532

Several comments are given below:

- To some extent, it is proved that the directional parametrised RRBF is capable of improving standard RRBF in terms of both training and test error.
- On average, the directional parametrised RRBF decreases 11% of the test error made by standard RRBF. However, it takes around 1 minute to run the program, while standard RRBF costs 4 seconds. Nevertheless, the time is still acceptable.
- In general, an significant increase in N_x does not effectively reduce test error. This might be one of the main differences between ESN and RRBF: For RRBF, N_x is usually chosen to be less then 15, while in ESN, N_x can be in hundreds, thousands, even in ten thousands. This structural difference may be part of the reason why RRBF has average test error which is 17% higher than that of ESN.

8.3.5 Unsupervised learning for RRBFN

There are many unsupervised learning algorithms for training RBF type neural networks. One of them is called Self Organising Map (SOM) (see Lukosevicius [2010]). See Figure (8.13).

Figure 8.13: Self-Organizing Maps

Its update equations are:

$$\mathbf{W}^i_{in}(n+1) = \mathbf{W}^i_{in}(n) + \eta(n)h(i,n)(\mathbf{U}(n) - \mathbf{W}^i_{in}(n)) \tag{8.3.9}$$
$$\mathbf{W}^i(n+1) = \mathbf{W}^i(n) + \eta(n)h(i,n)(\mathbf{X}(n) - \mathbf{W}^i(n)) \tag{8.3.10}$$

where $\eta(n)$ is the learning rate and $h(i,n)$ is called learning gradient distribution function:

$$h(i,n) = exp(\frac{-d_h(i, bmu(n))^2}{b_h(n)^2}) \tag{8.3.11}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In the above equation, bmu(n)=argmax$_i$($\mathbf{x}_i$(n)) is the index of the 'best matching unit'(BMU), $d_h(i,j)$ is the distance between units with indices i and j in the additionally defined topology for reservoir units. In our case, $d_h(i,j)$ is:

- either the Euclidean distance between $\mathbf{W}_{in}^i$and $\mathbf{W}_{in}^{bmu(n)}$.
- or the Euclidean distance between $\mathbf{W}_i$ and $\mathbf{W}_{bmu(n)}$.

Practically, every time a new input $\mathbf{U}(n)$ is imported, firstly the reservoir state $\mathbf{X}(n)$ is updated using $\mathbf{U}(n)$ and $\mathbf{X}(n-1)$:

$$\tilde{\mathbf{X}}_i(n) = exp\big(-\alpha||\mathbf{W}_{in}^i - \mathbf{U}(n)||^2 - \beta||\mathbf{W}^i - \mathbf{X}(n-1)||^2\big), \quad i = 1, ..., N_x \tag{8.3.12}$$

Then, Equations (8.3.9) and (8.3.10) are used to update rows of $\mathbf{W}_{in}$ and $\mathbf{W}$ adaptively. Meanwhile, this updating effect is controlled by a scaler called *learning rate* η. Empirical values for learning rate are $0.1, 0.01, 0.001$. In some cases, under certain criteria, the value of learning rate itself can dynamically change during the training process to better fit the dataset. But for simplicity, we use fixed value for η here.

Take Equation (8.3.9) for instance: consider input weight matrix $\mathbf{W}_{in} \in \mathbb{R}^{N_x \times N_u}$, with every row being a *center* vector to the new input $\mathbf{U}(n)$, then there are in total N_x center vectors. Hence, N_x is supposed to be the estimated number of *independent* center vectors. Ideally,

- These center vectors should be far from each other in Euclidean distance.
- Every new input $\mathbf{U}(n) \in \mathbb{R}^{N_u}$ should close to all of the center vectors, i.e. all the rows of $\mathbf{W}_{in}$.

However, due to the fact that $\mathbf{W}_{in}$ and $\mathbf{W}$ are randomly initialised, their rows may not function as suitable center vectors that perfectly match input dataset. Therefore, they need to be adaptively updated during the training process.

Whenever a new input $\mathbf{U}(n)$ is imported, its distances to every rows of $\mathbf{W}_{in}$ are computed. Among all N_x distances, the best matching unit (BMU) is the row index of $\mathbf{W}_{in}$ which is closest to $\mathbf{U}(n)$. Then, assuming $bmu = k$, the distances from the kth row to every other rows of $\mathbf{W}_{in}$ are calculated and the results are summarised in the term $h(i,n)$, where i means ith row and n indicates the time point. Specifically, it has the following expression:

$$h(i,n) = exp\big(\frac{-||\mathbf{W}_{in}^i(n) - \mathbf{W}_{in}^k(n)||^2}{(2\sigma^2)}\big)$$

Here the setting of σ varies from case to case, usually it is the volatility of the variable of interest.

Now, look back to Equation (8.3.9): if $\mathbf{U}(n)$ is closest to a center $\mathbf{W}_{in}^k(n)$, then $\mathbf{W}_{in}^k(n)$ becomes $\mathbf{W}_{in}^{bmu}(n)$. If $\mathbf{W}_{in}^{bmu}(n)$ is far from other center vectors, then the value of $h(i,n)$ should be small since the norm $||\mathbf{W}_{in}^i(n) - \mathbf{W}_{in}^{bmu}(n)||$ would be large. At the same time, if $\mathbf{U}(n)$ does not deviate too much from all the center vectors, $\mathbf{U}(n) - \mathbf{W}_{in}^i(n)$ would be relatively small for all i. Hence, the term $\eta(n)h(i,n)(\mathbf{U}(n) - \mathbf{W}_{in}^i(n))$ as a whole would be small if the learning rate $\eta(n)$ is selected and fixed. As a matter of fact, this is a desirable situation and $\mathbf{W}_{in}$ does not need to be modified much.

Moreover, it holds true that for $i = 1...N_x$,

$$||\mathbf{U}(n) - \mathbf{W}_{in}^{bmu}(n)|| <= ||\mathbf{U}(n) - \mathbf{W}_{in}^i(n)|| \tag{8.3.13}$$

$$1 = h(bmu(n), n) >= h(i,n) > 0 \tag{8.3.14}$$

However, it is not clear if $h(bmu(n),n)(\mathbf{U}(n) - \mathbf{W}_{in}^{bmu}(n)) >= h(i,n)(\mathbf{u}(n) - \mathbf{W}_{in}^i(n))$. That is, we are not aware if $\mathbf{W}_{in}^{bmu}$ moves greater distance towards $\mathbf{U}(n)$ than other rows do. Practically, it turns out that most of the time $\mathbf{W}_{in}^{bmu}$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

tends to move faster due to the exponential decay of $h(i, n)$. Nevertheless, there exist other models such as Neural Gas (NG) (see Lukosevicius [2010]) that require $h(bmu(n), n) = 0$. The above explanation similarly applies to Equation (8.3.10). Only the 'distance' involved changes.

Furthermore, due to the fact that the rows of $\mathbf{W}_{in}$ is always compared to $\mathbf{U}(n)$, in some cases the first N_x training inputs $\mathbf{U}(1)$, $\mathbf{U}(2)$,...,$\mathbf{U}(N_x)$ are used to initialise $\mathbf{W}_{in}$:

$$\mathbf{W}_{in} = \begin{bmatrix} — & U(1)^\top & — \\ — & U(2)^\top & — \\ & \vdots & \\ — & U(N_x)^\top & — \end{bmatrix} \in \mathbb{R}^{N_x \times N_u}$$

8.3.5.1 Pseudo code

The following is the pseudo code for training RRBFN with SOM:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 20 RRBFN with SOM Training Algorithm

Function$rrbfSOMTrain\mathbf{U}, \mathbf{y}, modelInputs$ $\{modelInputs \leftarrow N_x, \mathbf{W}_{in}, \mathbf{W}, \mathbf{X}_0$ *etc.*$\}$
$\mathbf{X} \in \mathbb{R}^{N_x \times nbDataPoints} \leftarrow$ *Design Matrix*
for $i = 1$ to $nbDataPoints$ **do**
  for $j = 1$ to N_x **do**
    $\mathbf{X}(j,i) = exp(-\alpha||[\mathbf{W}_{in}(j,:)]^T - \mathbf{U}(:,i)||^2 - \beta||[\mathbf{W}(j,:)]^T - \mathbf{X}(:,i-1)||^2)$; {*Finally update* $\mathbf{x}(n)$}
  end for
  $\mathbf{z}(:,i) = [1; \mathbf{U}(:,i); \mathbf{X}(:,i)]$; {*Vertical Concatenation*}
  $BMU_1 = ||\mathbf{W}_{in}(1,:)^T - \mathbf{U}(:,i)||$;
  $Index_1 = 1$;
  for $j = 2$ to N_x **do**
    $Test = ||\mathbf{W}_{in}(j,:)^T - \mathbf{U}(:,i)||$; {*To find* BMU_1}
    if $Test < BMU_1$ **then**
      $BMU_1 = Test$;
      $Index_1 = j$;
    end if
  end for
  $BMU_2 = ||\mathbf{W}(1,:)^T - \mathbf{X}(:,i)||$;
  $Index_2 = 1$;
  for $j = 2$ to N_x **do**
    $Test = ||\mathbf{W}(j,:)^T - \mathbf{X}(:,i)||$; {*To find* BMU_2}
    if $Test < BMU_2$ **then**
      $BMU_2 = Test$;
      $Index_2 = j$;
    end if
  end for
  for $k = 1$ to N_x **do**
    $\mathbf{W}_{in}(k,:) = \mathbf{W}_{in}(k,:) + \eta([\mathbf{U}(:,i)]^\top - \mathbf{W}_{in}(k,:))exp(-||\mathbf{W}_{in}(k,:) - \mathbf{W}_{in}(Index_1,:)||^2/(2\sigma^2))$; {*Update* $\mathbf{W}_{in}$ *and* $\mathbf{W}$}
    $\mathbf{W}(k,:) = \mathbf{W}(k,:) + \eta([\mathbf{x}(:,i)]^\top - \mathbf{W}(k,:))exp(-||\mathbf{W}(k,:) - \mathbf{W}(Index_2,:)||^2/(2\sigma^2))$;
  end for
end for
$m \leftarrow nbPointsToIgnore + 1$;
$S = \mathbf{z}(:, m : end)$;
$D = \mathbf{y}(:, m : end)$;
$\mathbf{W}_{out} = DS^T pinv(SS^T + \lambda I)$; {*Ridge Regression, pinv is pseudo inverse*}
EndFunction

8.3.5.2 Empirical results

Fix the optimal parameter setting for standard RRBF on stock 'MMM',

$[\alpha,\beta]$	[0.6,1]	Spectral Radius	1.25
Reservoir size N_x	12	Normalization Range	$[0,1]$
Reservoir Size $\mathbf{W}_{in}$	$[-1,1]$	Regularisation Term	10^{-8}

and only add unsupervised update equation for $\mathbf{W}_{in}$ and $\mathbf{W}$ (see Equations (8.3.9) and (8.3.10)) with learning rate $\eta = 0.01$ to the algorithm, the corresponding errors are shown below:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

	Training error(RMS)	Test error(RMS)
Linear Regression	0.00472	0.0036
Standard RRBF	0.00290	0.00173
Standard RRBF with SOM	0.00258	0.00163

Seen from the table, with adjustment of $\mathbf{W}_{in}$ and $\mathbf{W}$, the SOM algorithm manages to reduce 6% of the test error made by standard RRBF. Please see the evolution of input weight matrix $\mathbf{W}_{in}$ below:

Figure 8.14: Evolution of $\mathbf{W}_{in}$ during training

Note that $\mathbf{W}_{in}$ has dimension $N_x \times N_u = 12 \times 1$. In the above figure, since $N_x = 12$, the x-axis has 12 discrete points representing 12 positions in the vector $\mathbf{W}_{in}$. For each position, the corresponding y-value is the element value on that position. In theory, there supposed to be 1200 curves, with each curve shown in unique color. However, most of the curves coincide with each other and one can hardly tell one from another. This figure indicates that $\mathbf{W}_{in}$ has not changed a lot during the process. With a closer look at the figure, the 3th, 5th and 12th element may be found to evolve a bit during the training process.
Note that it is only in the training process that SOM is applied, the test set is always left untouched. At the end of the training process, the evolved $\mathbf{W}_{in}$ and $\mathbf{W}$ are used on test set. In order to reduce the chance of overfitting, no further evolution is undertaken at test stage.

Similar to $\mathbf{W}_{in}$, the evolution of $\mathbf{W}$ is also of interest. However, since $\mathbf{W}$ has $2D$ dimension of $N_x \times N_x$, it is difficult to visualise its adaptation process. Nevertheless, the initial and final formations of the reservoir weight matrix $\mathbf{W}$ are plotted:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 8.15: Initial W

Figure 8.16: Evolved W

Note that the deeper the colour, the higher the value. In the initial $\mathbf{W}$, the matrix is supposed to be sparse so that only 40% of them are non-zero. Comparing two figures, it could be seen that only 2nd and 11th row changes by a meaningful amount. This is partly because originally $\mathbf{W}$ is already relative close to optimal setting.

Now, keeping the other parameters fixed, if the range of $\mathbf{W}_{in}$ changes to $[-0.5, 0.5]$, then the errors generated are as follows:

	Training error(RMS)	Test error(RMS)
Linear Regression	0.00472	0.0036
Standard RRBF	0.00332	0.00213
Standard RRBF with SOM	0.00319	0.00189

The corresponding evolution of $\mathbf{W}_{in}$ is also displayed:

Figure 8.17: Evolution of $\mathbf{W}_{in}$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Several comments are now presented:

- Seen from the table, SOM successfully reduces 11% of the error made by standard RRBF. The error reduction doubles compared to the previous case.
- The figure shows a relative large evolution on the 3th, 5th and 12th rows. It looks as if SOM tries to 'rectify' these rows of W_{in} and lead them to the right track. Generally, in those much more complex tasks, where optimal parameter settings are hard to find, SOM can guide the system adaptively in the right direction.

Now, the general performance of SOM is tested on 100 stocks. The parameter settings are kept exactly the same as the previous one used for standard RRBF tests across 100 stocks. The resultant errors are as follows:

	Average Training error(RMS)	Average Test error(RMS)
Linear Regression	0.00809	0.00799
Standard RRBF	0.00531	0.00598
Standard RRBF with SOM	0.00509	0.00571

Seen from the table, on average SOM reduces test errors by 5%, which is half of the previous 11% reduction on a specific stock. Moreover, the error percentage reduction distribution across 100 stocks is also displayed:

Figure 8.18: Histogram of Error Reduction

Since a positive number means that SOM does reduce test errors, statistically it turns out that 72% of the time SOM successfully decreases test errors made by standard RRBF, but in the rest 28% of the time, SOM undesirably increases the errors. The failure of SOM to decrease test errors can be partly due to a common setting of the learning rate η. Throughout the process only one single η is used. It is fixed during the training process, and over stocks. Therefore, if a changeable η is used, it is likely that SOM will remain effective more frequently. The following is a prediction comparison on a sample stock:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 8.19: RRBF on test set

Figure 8.20: RRBF with SOM on test set

As for directional parametrised RRBF, the same experiments are done on the same 100 stocks using a common parameter setting. The results show that despite that on average training errors decrease by 6%, only 44% of the time

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

SOM manages to reduce the test error made by directional parametrised RRBF. Moreover, the training process takes much more time than previous experiments. The incompatibility of SOM to directional parametrised may be due to that directional parametrised RRBF itself has already scaled element-wise the distance between $\mathbf{U}(n)$ and $\mathbf{W}_{in}^{i}$, hence a further modification on $\mathbf{W}_{in}$ might create adverse effects.

Overall, the SOM model is promising and useful on standard RRBF in the sense that most of the time it is able to adaptively lead rows of $\mathbf{W}_{in}$ and $\mathbf{W}$ to the right positions when the original settings are not suitable for input dataset. Moreover, if the learning rate could be made flexible, or another $h(i, n)$ was utilised, chances are there for SOM to be further developed.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 9

Deep learning

Deep learning is part of a broader family of machine learning methods based on artificial neural networks (see Chapter (5)) with representation learning that can be supervised, semi-supervised or unsupervised. A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. DNNs are typically feedforward networks in which data flows from the input layer to the output layer without looping back. At first, the DNN creates a map of virtual neurons and assigns random numerical values, or weights, to connections between them. The weights and inputs are multiplied and return an output between 0 and 1. If the network does not accurately recognise a particular pattern, an algorithm would adjust the weights. That way the algorithm can make certain parameters more influential, until it determines the correct mathematical manipulation to fully process the data. Deep learning systems have a substantial credit assignment path (CAP) depth representing the chain of transformations from input to output. CAPs describe potentially causal connections between input and output. For a feedforward neural network, the depth of the CAPs is that of the network and is the number of hidden layers plus one (as the output layer is also parameterised). For recurrent neural networks, in which a signal may propagate through a layer more than once, the CAP depth is potentially unlimited. Most researchers agree that deep learning involves CAP depth higher than two. Deep models with a CAP greater than two are able to extract better features than shallow models and hence, extra layers help in learning the features effectively.
While some deep learning models comprise recurrent neural networks (RNNs) and convolutional neural networks (CNNs), they can also include propositional formulas or latent variables organised layer-wise in deep generative models such as the nodes in deep belief networks and deep Boltzmann machines. Deep learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs.
As with ANNs, many issues can arise with naively trained DNNs. Some common issues are overfitting, training difficulties, and computation time.

Deep neural networks have found applications in many real-life tasks. In particular, the field of computer vision has hugely benefited from deep neural networks for various applications ranging from pose estimation, segmentation, action recognition, face recognition, among others. Recently, deep networks focussed on learning directly from raw data without feature decorrelation, feature selection, image restoration and denoising, or with very minimal data processing. For instance, the MNIST dataset (see Section (1.6.1.2)) is no longer considered as a hard dataset. Tackling harder learning tasks with many layers has motivated the development of very deep networks (see Srivastava et al. [2015] [2015b]). Empirical evidences showed that increasing the depth of deep models increases the complexity of the functions that they can capture (see He et al. [2016]). It is assumed that with the raise of big data (see Section (3.2.4)), we can learn a robust classifier that can generalise well on unseen data. As a result, deep networks with hundreds of layers developed (see Shen et al. [2016]). However, very deep networks are difficult to train, especially

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

learning the identity function. Some approaches relied on some means of routing untransformed features from the earlier layers through the model, that is, learning near identity functions for the different layers (see Srivastava et al. [2015b], He et al. [2016], Huang et al. [2017b]). However, this learning setup favours learning untransformed features, which may impact model generalisation. Oyedotun et al. [2020] proposed two variants of highway network, HWGC and HWCC, employing feature summation and concatenation respectively.

See textbooks by Bishop [2006], Bengio [2009], Montavon et al. [2012], Deng et al. [2014], Goodfellow et al. [2016], among others.

9.1 Some deep neural networks

9.1.1 Convolutional neural network

See textbooks by Aghdam et al. [2017], Lu et al. [2017], lecture notes by Li et al. [2017]. Most figures are taken from these lecture notes.

9.1.1.1 An overview

9.1.1.1.1 Description A convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analysing visual imagery (see Section (1.4)). They successively convolve the input image with learned filters to build up a hierarchy of feature maps. This hierarchical approach allows to learn more complex, as well as translation and distortion invariant, features in higher layers. As a result, CNNs are used to learn a hierarchy of features which are used for classification purposes. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, and natural language processing. CNNs are regularised versions of multilayer perceptrons (MLP) (see Section (5.1.3)). As such, the network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, RELU layer, that is, activation function (see Section (10.1.2)), pooling layers, fully connected layers and normalisation layers. Mathematically, the process is a cross-correlation rather than a convolution (see Appendix (23.2.3)). It has significance for the indices in the matrix, and thus which weights are placed at which index.

Due to the full connectivity between nodes (see Section (5.2)), MLP suffer from the curse of dimensionality and thus do not scale well to higher resolution images. For example, in CIFAR-10, images are only of size $32 \times 32 \times 3$ (32 wide, 32 high, 3 colour channels), so a single fully connected neuron in a first hidden layer of a regular neural network would have $32 \times 32 \times 3 = 3,072$ weights. This is illustrated in Figure (9.1). However, a 200×200 image, would lead to neurons that have $200 \times 200 \times 3 = 120,000$ weights. Further, such network architecture does not take into account the spatial structure of data, ignoring locality of reference in image data, both computationally and semantically. Thus, full connectivity of neurons is wasteful for purposes such as image recognition that are dominated by spatially local input patterns. CNNs mitigate the challenges posed by the MLP architecture by exploiting the strong spatially local correlation present in natural images.

Figure 9.1: CNN connected layer

As opposed to MLPs, CNNs have the following distinguishing features:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- 3D volumes of neurons. The layers of a CNN have neurons arranged in three dimensions: width, height and depth. See Figure (9.2) in the case of our $32 \times 32 \times 3$ image. The neurons inside a layer are connected to only a small region of the layer before it, called a receptive field (RF). Distinct types of layers, both locally and completely connected, are stacked to form a CNN architecture.
- Local connectivity: Following the concept of receptive fields, CNNs exploit spatial locality by enforcing a local connectivity pattern between neurons of adjacent layers. The architecture thus ensures that the learned Filters produce the strongest response to a spatially local input pattern. Stacking many such layers leads to non-linear filters that become increasingly global (responsive to a larger region of pixel space) so that the network first creates representations of small parts of the input, then from them assembles representations of larger areas.
- Shared weights: In CNNs, each filter is replicated across the entire visual field. These replicated units share the same parameterisation (weight vector and bias) and form a feature map. This means that all the neurons in a given convolutional layer respond to the same feature within their specific response field. Replicating units in this way allows for features to be detected regardless of their position in the visual field, thus constituting a property of translation invariance.

Figure 9.2: 3D image and filter

Together, these properties allow CNNs to achieve better generalisation on vision problems. Each layer learns to transform its input data into a slightly more abstract and composite representation. For instance, the first representational layer may abstract the pixels and encode edges; the second layer may compose and encode arrangements of edges; the third layer may encode a nose and eyes; and the fourth layer may recognise that the image contains a face. Weight sharing dramatically reduces the number of free parameters learned, thus lowering the memory requirements for running the network and allowing the training of larger, more powerful networks.

9.1.1.1.2 Some history Inspired by the work of Hubel and Wiesel in the 1950s and 1960s (see Hubel et al. [1968]) who showed that cat and monkey visual cortexes contain neurons that individually respond to small regions of the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

visual field, Fukushima [1980] introduced the neocognitron. This is illustrated in Figure (5.3). It introduced the two basic types of layers in CNNs:

1. Convolutional layers: They contain units whose receptive fields cover a patch of the previous layer. The weight vector (the set of adaptive parameters) of such a unit is often called a filter. Units can share filters.

2. Downsampling layers: They contain units whose receptive fields cover patches of previous convolutional layers. Such a unit typically computes the average of the activations of the units in its patch. This downsampling helps to correctly classify objects in visual scenes even when the objects are shifted.

Several supervised and unsupervised learning algorithms have been proposed over the decades to train the weights of a neocognitron. In general, the CNN architecture is usually trained through backpropagation (see Section (5.3)).
Rumelhart et al. [1986b] proposed a CNN based on weight sharing as a regularisation technique. Waibel [1987] introduced the time delay neural network (TDNN) which was the first convolutional network, as it achieved shift invariance by utilising weight sharing in combination with backpropagation training. TDNNs are convolutional networks that share weights along the temporal dimension. They allow speech signals to be processed time-invariantly. This inspired translation invariance in image processing with CNNs.
Zhang et al. [1988] proposed a shift invariant neural network for image character recognition. Zhang et al. [1990] modified the architecture and training algorithm. The results indicated that the processing was effective and the network could deal with the shifted or distorted patterns. The model was applied for medical image processing and automatic detection of breast cancer in mammograms.
Denker et al. [1989] considered a system that recognises hand-printed digits, using a combination of classical techniques and neural-net methods. The system has been trained and tested on real-world data, derived from zip codes seen on actual U.S. Mail. It involved convolutions in which the kernel coefficients had been laboriously hand designed.
LeCun et al. [1989] integrated constraints into a back-propagation network through the architecture of the network. They used back-propagation to learn the convolution kernel coefficients directly from images of hand-written numbers. The learning was thus fully automatic, performed better than manual coefficient design, and was suited to a broader range of image recognition problems and image types.
In order to classify digits, LeCun et al. [1998] proposed LeNet-5, a pioneering 7-level convolutional network, that was applied by to recognise hand-written numbers on checks (British English: cheques) digitised in 32×32 pixel images. The ability to process higher resolution images requires larger and more layers of convolutional neural networks, so this technique is constrained by the availability of computing resources. Figure (9.3) illustrates the mechanism of the LeNet-5 network.

Figure 9.3: CNN LeNet-5

Oh et al. [2004] showed that standard neural networks could be greatly accelerated on GPUs. Their implementation was 20 times faster than an equivalent implementation on CPU. Steinkraus et al. [2005] also emphasised the value of GPGPU for machine learning. At IDSIA, Ciresan et al. [2010] showed that even deep standard neural networks

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

with many layers could be quickly trained on GPU by supervised learning through the old method known as back-propagation. Ciresan et al. [2011] extended this GPU approach to CNNs, achieving an acceleration factor of 60, with impressive results.
Recent success of CNNs can be found in Hinton et al. [2012], Ciresan et al. [2012], Krizhevsky et al. [2017].

9.1.1.2 Modelling images

9.1.1.2.1 Biological nervous systems In biological processes, the connectivity pattern between neurons resembles the organisation of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field (RF). Alonso et al. [2008] defined the receptive field as a portion of sensory space that can elicit neuronal responses when stimulated. The sensory space can be defined in a single dimension (such as carbon chain length of an odorant), two dimensions (such as skin surface) or multiple dimensions (such space, time and tuning properties of a visual receptive field). The neuronal response can be defined as firing rate (number of action potentials generated by a neuron) or include also subthreshold activity (depolarisations and hyperpolarisations in membrane potential that do not generate action potentials). When used in CNN, the term adopts a meaning reminiscent of receptive fields in actual biological nervous systems. In that setting, RF is the region of visual space within which visual stimuli affect the firing of a single neuron. Neighbouring cells have similar and overlapping receptive fields. RFs of different neurons partially overlap such that they cover the entire visual field. Thus, each convolutional neuron processes data only for its receptive field.

9.1.1.2.2 The convolution operation Since CNNs are used primarily in the field of computer vision, the data that the neurons represent is typically an image. Each input neuron represents one pixel from the original image. In that setting, a very high number of neurons would be necessary to learn features as well as classify data with fully connected feedforward neural networks. This is due to the very large input sizes associated with images, where each pixel is a relevant variable. For instance, a fully connected layer for a (small) image of size 100×100 has $10,000$ weights for each neuron in the second layer.
In a CNN, the first layer of neurons is composed of all the input neurons; neurons in the next layer will receive connections from some of the input neurons (pixels), but not all, as in a MLP. Hence, instead of having each neuron receive connections from all neurons in the previous layer, CNNs use a receptive field-like layout in which each neuron receives connections only from a subset of neurons in the previous (lower) layer. The RF of a neuron in one of the lower layers encompasses only a small area of the image, while the RF of a neuron in subsequent (higher) layers involves a combination of RFs from several (but not all) neurons in the layer before. That is, a neuron in a higher layer looks at a larger portion of the image than does a neuron in a lower layer. In this way, each successive layer is capable of learning increasingly abstract features of the original image.
The convolution operation reduces the number of free parameters, allowing the network to be deeper with fewer parameters. For instance, regardless of image size, tiling regions of size 5×5, each with the same shared weights, requires only 25 learnable parameters. In this way, it resolves the vanishing or exploding gradients problem in training traditional multi-layer neural networks with many layers by using backpropagation.

9.1.1.3 The CNN architecture

A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume through a differentiable function. We briefly introduce some of these layers.

9.1.1.3.1 Convolutional layers Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. The parameters of the layer consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. Each convolutional layer within the network should have the following attributes:

- Input is a tensor with shape (number of images) $\times$ (image width) $\times$ (image height) $\times$ (image depth).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Number of convolutional kernels.
  - Width and height of kernels are hyper-parameters.
  - Depth of kernels must be equal to the image depth.

During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. This is illustrated Figure (9.4) where the activation map is a 28×28 sheet of neuron outputs. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input. Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. This is illustrated in Figure (9.5) in the case of six 5×5 filters. We get a new image of size $28 \times 28 \times 6$. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

Figure 9.4: CNN convolution

Figure 9.5: CNN convolution with 6 separate activation maps

9.1.1.3.2 Pooling Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters (sub-region) at one layer into a single neuron in the next layer. See Figure (9.6). CNNs may include local or global

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

pooling layers. Local pooling combines small clusters, typically 2×2. Global pooling acts on all the neurons of the convolutional layer.
Intuitively, the exact location of a feature is less important than its rough location relative to other features. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters, memory footprint and amount of computation in the network, and hence to also control overfitting.
Thus, pooling is a form of non-linear down-sampling. Even though there are several non-linear functions, in general, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer (see Ciresan et al. [2012]). Scherer et al. [2010] showed that max pooling got faster convergence during training. Average pooling uses the average value from each of a cluster of neurons at the prior layer. Note, Region of Interest pooling (also known as RoI pooling) is a variant of max pooling, in which output size is fixed and input rectangle is a parameter. Further, both max and average pooling can be applied by using overlapping windows of size $2p \times 2p$ placed q units apart. Then the windows overlap if $q < p$. It reduces the chance of overfitting the training set.

Figure 9.6: CNN pooling layer

9.1.1.3.3 Receptive field In neural networks, each neuron receives input from some number of locations in the previous layer. In a fully connected layer, each neuron receives input from every element of the previous layer. However, when dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume because such a network architecture does not take the spatial structure of the data into account. Convolutional networks exploit spatially local correlation by enforcing a sparse local connectivity pattern between neurons of adjacent layers. Neurons receive input from only a restricted sub-area of the previous layer (input volume). Typically, the sub-area is of a square shape (such as size 5 by 5). The input area of a neuron is called its receptive field (RF). So, in a fully connected layer, the receptive field is the entire previous layer, while in a convolutional layer, it is smaller than the entire previous layer. The connections are local in space (along width and height), but always extend along the entire depth of the input volume. Such an architecture ensures that the learnt filters produce the strongest response to a spatially local input pattern.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

9.1.1.3.4 Weights Each neuron in a neural network (NN) computes an output value by applying some function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is specified by a vector of weights and a bias (typically real numbers). Learning in a NN progresses by making incremental adjustments to the biases and weights (see Section (5.1.4)). The vector of weights and the bias are called a filter and represents some feature of the input (such as a particular shape). A distinguishing feature of CNNs is that many neurons share the same filter. This reduces memory footprint because a single bias and a single vector of weights is used across all receptive fields sharing that filter, rather than each receptive field having its own bias and vector of weights.

9.1.1.3.5 Spatial arrangement We consider a volume of size $W_1 \times H_1 \times D_1$ with width W_1, height H_1, and depth D_1. We are interested in the new volume with size $W_2 \times H_2 \times D_2$. This is illustrated in Figure (9.7) where the initial volume is of size $32 \times 32 \times 3$ and the new volume is of size $28 \times 28 \times 5$.

Figure 9.7: CNN convolution volume

There are three hyperparameters controlling the size of the output volume of the convolutional layer: the depth, stride S and zero-padding P.

- The depth of the output volume controls the number of neurons in a layer that connect to the same region of the input volume. These neurons learn to activate for different features in the input. For example, if the first convolutional layer takes the raw image as input, then different neurons along the depth dimension may activate in the presence of various oriented edges, or blobs of colour.
- Stride controls how depth columns around the spatial dimensions (width and height) are allocated. When the stride is 1 then we move the filters one pixel at a time, leading to heavily overlapping receptive fields between the columns, and also to large output volumes. When the stride is 2 then the filters jump 2 pixels at a time as they slide around. Similarly, for any integer $S > 0$, a stride of S causes the filter to be translated by S units at a time per output. However, stride lengths of $S = 3$ are rare. It is illustrated in Figure (9.8) with 7×7 input (spatially) and 3×3 filter. Here S could be 1 giving 5×5 output, or 2 giving 3×3 output. However, we can not have $S = 3$ since it would not fit. The receptive fields overlap less and the resulting output volume has smaller spatial dimensions when stride length is increased.
- Sometimes it is convenient to pad the input with zeros on the border of the input volume. It is illustrated in Figure (9.9) with pad with one pixel border. The size of this padding is a third hyperparameter. Padding provides control of the output volume spatial size. In particular, sometimes it is desirable to exactly preserve the spatial size of the input volume.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 9.8: CNN stride

Figure 9.9: CNN zero pad

The spatial size W_2 of the output volume can be computed as a function of the input volume size W_1, the kernel field (or filter) size of the convolutional layer neurons K, the stride with which they are applied S, and the amount of zero padding P used on the border. The formula for calculating how many neurons fit in a given volume is given by

$$\frac{W_1 - K + 2P}{S} + 1$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, in the special case where we do not have zero padding, we set $P = 0$ in the above formula. Width and height are computed equally by symmetry. Further, $D_2 = K$.

9.1.1.4 Mathematical representation

We consider a $(L+1)$-layer perceptron with D input units, C output units, and several hidden units (see Section (5.1.3.2)). The units are arranged in layers such that the multilayer perceptron comprises an input layer, an output layer and L hidden layers. The ith unit within layer l computes the output

$$y_i^{(l)} = f\left(z_i^{(l)}\right) \text{ with } z_i^{(l)} = \sum_{k=1}^{m^{(l-1)}} w_{i,k}^{(l)} y_k^{(l-1)} + w_{i,0}^{(l)}$$

where $w_{i,k}^{(l)}$ denotes the weighted connection from the kth unit in layer $(l-1)$ to the ith unit in layer l, and $w_{i,0}^{(l)}$ is an external input to the unit referenced as a bias. $m^{(l)}$ denotes the number of units in layer l, so that $D = m^{(0)}$ and $C = m^{(L+1)}$. We consider the bias as a weight and introduce the dummy unit $y_0^{(l)}$. In each layer we get

$$z_i^{(l)} = \sum_{k=0}^{m^{(l-1)}} w_{i,k}^{(l)} y_k^{(l-1)} \text{ or } z^{(l)} = w^{(l)} y^{(l)} \tag{9.1.1}$$

where $z^{(l)}$ is the input vector, $w^{(l)}$ is the weight matrix. The multilayer perceptron is given by the function $y(\cdot, w) : \mathbb{R}^D \to \mathbb{R}^C$ where the output vector $y(x, w)$ comprises the output values $y_i(x, w) = y_i^{(L+1)}$.
We briefly introduce multidimensional discrete convolution (see Appendix (23.2.3)) in the case of image processing. We consider the grayscale image defined by the function

$$I : \{1, ..., N_1\} \times \{1, ..., N_2\} \to W \subseteq \mathbb{R}\,,\ (i, j) \to I(i, j)$$

Thus, the image is represented by an array of $m_1 \times m_2$. Given the filter $K \in \mathbb{R}^{(2r_1+1)\times(2r_2+1)}$, the 2-dimensional discrete convolution of the image I with the filter K is given by

$$(I * K)(n_1, n_2) = \sum_{k_1=-r_1}^{r_1} \sum_{k_2=-r_2}^{r_2} K(k_1, k_2) I(n_1 + k_1, n_2 + k_2)$$

where the filter K is given by

$$K = \begin{pmatrix} (-r_1, -r_2) & \cdots & (-r_1, r_2) \\ \vdots & \vdots & \vdots \\ (r_1, -r_2) & \cdots & (r_1, r_2) \end{pmatrix}$$

***Remark* 9.1.1** *The behaviour of the multidimensional discrete convolution towards the border of the image needs to be properly defined.*

See Appendix (23.2.3) for a description of the Gaussian filter which leads to the Gaussian convolution.
We now present a few layers commonly used in CNNs (see Jarrett [2009]) which serve as building block to obtain more complex architectures used for classification (see Krizhevsky et al. [2017]).

9.1.1.4.1 Convolutional layer We let l be a convolutional layer with input comprising $m_1^{(l-1)}$ feature (or activation) maps from the previous layer, each of size $m_2^{(l-1)} \times m_3^{(l-1)}$. In the special case where $l = 1$, the input is a single image I consisting of one or more channels. Thus, the CNN directly accepts raw images as input. The output of that layer consists of $m_1^{(l)}$ feature maps of size $m_2^{(l)} \times m_3^{(l)}$. This is illustrated in Figure (9.10).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 9.10: Single convolution layer.

We let $Y_i^{(l)}$ be the ith feature map in layer l given by

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K^{(l)}(i,j) * Y_j^{(l-1)} \tag{9.1.2}$$

where $B_i^{(l)}$ is a bias matrix and $K^{(l)}(i,j)$ is the filter of size $(2r_1^{(l)}+1)\times(2r_1^{(l)}+1)$ connecting the jth feature map in layer $(l-1)$ with the ith feature map in layer l. From Remark (9.1.1), $m_2^{(l)}$ and $m_3^{(l)}$ are influenced by border effects. Thus, focusing on the valid region of the input feature maps, the sizes of the output feature maps are

$$m_2^{(l)} = m_2^{(l-1)} - 2r_1^{(l)} \text{ and } m_3^{(l)} = m_3^{(l-1)} - 2r_2^{(l)}$$

In general, the filters used for computing a fixed feature map $Y_i^{(l)}$ are the same. That is, $K^{(l)}(i,j) = K^{(l)}(i,k)$ for $j \neq k$. Further, the sum in the computation of $Y_i^{(l)}$ may run over a subset of the input feature maps. The feature map $Y_i^{(l)}$ in layer l in Equation (9.1.2) can be related to the multilayer perceptron. The unit at position (n_1,n_2) computes the output

$$\begin{aligned}
\left(Y_i^{(l)}\right)_{n_1,n_2} &= \left(B_i^{(l)}\right)_{n_1,n_2} + \sum_{j=1}^{m_1^{(l-1)}} \left(K^{(l)}(i,j) * Y_j^{(l-1)}\right)_{n_1,n_2} \\
&= \left(B_i^{(l)}\right)_{n_1,n_2} + \sum_{j=1}^{m_1^{(l-1)}} \sum_{k_1=-r_1}^{r_1} \sum_{k_2=-r_2}^{r_2} \left(K^{(l)}(i,j)\right)_{k_1,k_2} \left(Y_j^{(l-1)}\right)_{n_1+k_1,n_2+k_2}
\end{aligned}$$

The trainable weights of the CNN are obtained in the filters $K^{(l)}(i,j)$ and the bias matrices $B_i^{(l)}$. Note, sub-sampling can be used to decrease the effect of noise and distortion (see Ciresan et al. [2011]). Given the skipping factors $s_1^{(l)}$ and $s_2^{(l)}$, we skip a fixed number of pixels, both in horizontal and in vertical direction, before applying the filter again. In that setting, the sizes of the output feature maps are

$$m_2^{(l)} = \frac{m_2^{(l-1)} - 2r_1^{(l)}}{s_1^{(l)}+1} \text{ and } m_3^{(l)} = \frac{m_3^{(l-1)} - 2r_2^{(l)}}{s_2^{(l)}+1}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

9.1.1.4.2 Non-linearity layer In the case where layer l is a non-linearity layer, its input is given by $m_1^{(l)}$ feature maps and its output comprises $m_1^{(l)} = m_1^{(l-1)}$ feature maps, each of size $m_2^{(l-1)} \times m_3^{(l-1)}$ such that $m_2^{(l)} = m_2^{(l-1)}$ and $m_3^{(l)} = m_3^{(l-1)}$, given by

$$Y_i^{(l)} = f\big(Y_i^{(l-1)}\big)$$

where f is the point-wise activation function used in layer l. Jarrett [2009] added a gain coefficient g_i, getting

$$Y_i^{(l)} = g_i f\big(Y_i^{(l-1)}\big)$$

In the special case where the activation function is a hyperbolic tangent, the layer is denoted by F_{CSG}.

9.1.1.4.3 Rectification If the layer l is a rectification layer R_{abs}, its input comprises $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$ and the absolute value for each component of the feature maps is given by

$$Y_i^{(l)} = \big|Y_i^{(l)}\big|$$

where the absolute value is computed point-wise such that the output consists of $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size. Jarrett [2009] showed that rectification plays a central role in achieving good performance.

9.1.1.4.4 Feature pooling and sub-sampling layer Reducing the resolution allows to reduce noise and distortions. Sub-sampling the feature maps can be obtained by applying skipping factors or it can combined with pooling in a separate layer (see LeCun et al. [2010]).
We let l be a pooling layer with output comprising $m_1^{(l)} = m_1^{(l-1)}$ feature maps of reduced size. Pooling operates by placing windows at non-overlapping position in each feature map and keeping one value per window such that the feature maps are sub-sampled.
The boxcar filter K_B of size $(2r_1+1) \times (2r_2+1)$ is an example of average pooling. It is given by

$$\big(K_B\big)_{n_1,n_2} = \frac{1}{(2r_1+1)(2r_2+1)}$$

9.1.1.4.5 Fully connected layer We let l be a fully connected layer. If layer $(l-1)$ is also a fully connected layer, we can apply Equation (9.1.1). Otherwise layer l expects $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$ as input and the ith unit in layer l is given by

$$y_i^{(l)} = f\big(z_i^{(l)}\big) \text{ with } z_i^{(l)} = \sum_{j=1}^{m_1^{(l-1)}} \sum_{n_1=1}^{m_2^{(l-1)}} \sum_{n_2=1}^{m_3^{(l-1)}} w_{i,j,n_1,n_2}^{(l)} \big(Y_j^{(l-1)}\big)_{n_1,n_2}$$

where $w_{i,j,n_1,n_2}^{(l)}$ is the weight connecting the unit at position (n_1,n_2) in the jth feature map of layer $(l-1)$ and the ith unit in layer l. While a fully connected layer already includes the non-linearities, in a convolutional layer the non-linearities are separated in their own layer.

9.2 Training deep networks

While the depth of neural networks is crucial for their success (see Yu et al. [2013], Simonyan et al. [2015]), training networks becomes more difficult as depth increases. As a results, several techniques developed to deal with the difficulties of training deep networks. For instance, Sutskever et al. [2013] showed that stochastic gradient descent (SGD), used with properly calibrated meta-parameters and a strong form of momentum, also works well on

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

very deep neural network optimisation problems. Martens et al. [2012] presented a report on the use of Hessian-Free optimisation (HF) in deep networks, which is an approach for unconstrained minimisation of real-valued smooth objective functions working by performing a sub-optimisation using the linear conjugate gradient algorithm (CG). Well-designed initialisation strategies are critical for training moderately deep networks. Examples include the normalised variance-preserving initialisation for certain activation functions (see Glorot et al. [2010], Kaiming et al. [2015]). Experiments showed that certain activation functions based on local competition may help to train deeper networks (see Goodfellow et al. [2013]). While skip connections between layers have long been used in ANNs, they have been considered for improving the flow of information (see Raiko et al. [2012], Graves [2013]). Inspired from LSTM, Srivastava et al. [2015] [2015b] proposed to modify the architecture of very deep feedforward networks with an adaptive gating mechanism that allows for computation paths along which information can flow across many layers without attenuation. These paths are called information highways yielding highway networks. Also inspired by the LSTM, Kalchbrenner et al. [2015] proposed a similar model. He et al. [2016] proposed Residual Network (ResNet) with skip connections of identity mappings for training very deep networks. Oyedotun et al. [2020] proposed two variants of highway network, HWGC and HWCC, employing feature summation and concatenation respectively.

9.2.1 Highway networks

A highway network is a neural architecture (a stack of highway blocks) enabling the training of extremely deep networks using simple stochastic gradient descent (see Srivastava et al. [2015b]). The highway block relies on a gating mechanism for routing untransformed lower layer features through the model to alleviate the problem of model optimisation.

9.2.1.1 Description

As discussed in Section (5.1.3), a plain feedforward neural network typically consists of $L \in \mathbb{N}$ layers where the lth layer ($l \in \{1, 2, ..., L\}$) applies a non-linear transformation H, parameterised by $W_{H,l}$, on its input x_l to produce its output y_l. Here, x_1 is the input to the network and y_L is the output of the network. For simplicity of exposition, omitting the layer index and biases, we get

$$y = H(x, W_H)$$

In the case of neural networks, at the lth layer, we get Equation (5.1.5)

$$H_l = \Phi^l \circ U^l$$

where the function $U^l : \mathbb{R}^{N_l} \to \mathbb{R}^{N_{l+1}}$ for $1 \leqslant l \leqslant L-1$ is the affine function, and the function $\Phi^l : \mathbb{R} \to \mathbb{R}$ is the activation function. That is, H_l is an affine transform followed by a non-linear activation function. Note, H may take other forms, such as convolutional or recurrent forms. A highway network, additionally defines two non-linear transformations T and C such that

$$y = H(x, W_H) \cdot T(x, W_T) + x \cdot C(x, W_H)$$

where T is referred to as the transform gate, and C as the carry gate, since they express how much of the output is produced by transforming the input and carrying it, respectively. The dot operator is used to denote element-wise multiplication. Note, T and C can take any form, but should map inputs to values in $(0, 1)$ to be interpreted as gates. Setting $C = 1 - T$, we get

$$y = H(x, W_H) \cdot T(x, W_T) + x \cdot \big(1 - T(x, W_T)\big)$$

For particular values of T we get

$$y = \begin{cases} x \text{ if } T(x, W_T) = 0 \\ H(x, W_H) \text{ if } T(x, W_T) = 1 \end{cases}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Similarly for the Jacobian of the layer transform

$$\frac{dy}{dx} = \begin{cases} I \text{ if } T(x, W_T) = 0 \\ H^{'}(x, W_H) \text{ if } T(x, W_T) = 1 \end{cases}$$

where I is the identity matrix and $H^{'}$ is the derivative of H with respect to x. Thus, depending on the output of the transform gates, the highway layer can smoothly vary its behaviour between that of a plain layer and that of a layer which simply passes its inputs through. Here, the ith block computes a block state $H_i(x)$ and a transform gate output $T_i(x)$, and it generates the block output

$$y_i = H_i(x) \times T_i(x) + x_i \times \big(1 - T_i(x)\big)$$

which is connected to the next layer.

9.2.1.2 Implementation

Given the input vector $x \in \mathcal{X}$ and output vector $y \in \mathcal{Y}$, we define the matrices $H = H(x, W_H)$, $T = T(x, W_T)$, and $C = C(x, W_C)$ where W_H, W_T, and W_C are the transformation parameters to be learned.
The forward pass is given by

$$y = H \cdot T + x \cdot C$$

Setting $C = 1 - T$, it simplifies to

$$y = H \cdot T + x \cdot (1 - T)$$

Note, the dimensionality of x, y, H and T must be the same for this equation to be valid. One solution is to use a plain layer, without highways, to change dimensionality. One can also replace the input vector x with $\widehat{x}$ obtained by suitably sub-sampling or zero-padding x. In the case of convolutional highway layers, weight-sharing and local receptive fields for both H and T are utilised. The authors used the same sized receptive fields for both, and zero-padding to make sure they match the input size.
During the backward pass, we only need to compute the derivatives of x, T, H as follows:

$$\begin{aligned} dH &= T \cdot dy \\ dT &= (H - x) \cdot dy \\ dx &= (1 - T) \cdot dy \end{aligned}$$

9.2.1.3 Shortcomings

Some shortcomings to the highway networks are:

- The highway blocks used for constructing the model makes it difficult to learn new feature transformations (or latent representations), which are important for generalisation as training progresses.
- Employing a gate for every layer translates to roughly doubling the number of parameters for the model. As a result, model overfitting can become a concern.

The first point is related to the way the highway blocks learn: The gating units use the sigmoid activation function limiting output range to $(0, 1)$. At the start of training, the biases of the gating units of all highway blocks are initialised to negative values, which position the gating units close to saturation. The gating units further go into saturation as training progresses. This learning setup favours learning essentially untransformed features, which may impact model generalisation that relies on learning new feature representations in the different hidden layers.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Part III

The mathematical formalism

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 10

Supervised learning

We have presented artificial and recurrent neural networks in Chapters (5) and (6), discussing the construction of neurons and introducing different networks with their respective algorithms. We are now going to give a more formal representation of such networks.

10.1 Introduction to learning theory

Vapnik [1982] developed a fundamental approach to learning theory called structural risk minimisation. We are going to introduce the main ideas.

10.1.1 Function estimation

10.1.1.1 Risk minimisation

In a general supervised learning problem we have two spaces of objects $\mathcal{X}$ and $\mathcal{Y}$ and would like to learn a function $h : \mathcal{X} \to \mathcal{Y}$ which outputs an object $y \in \mathcal{Y}$, given $x \in \mathcal{X}$. The function h is sometime called a **score function** or **hypothesis**. It is an element of some space of possible functions H called the hypothesis space. We assume a given sample $\{(x_1, y_1), ..., (x_n, y_n)\}$ where $x_i \in \mathcal{X}$ is an input and $y_i \in \mathcal{Y}$ is the corresponding response that we wish to get from $h(x_i)$. We consider probabilistic learning models (see Chapter (11)), and assume that there is a joint probability distribution $P(x, y)$ (see Definition (24.1.10)) and that the sample is made of independent and identically distributed (i.i.d.) variables drawn from that distribution. That is, we model uncertainty (such as noise in data) by letting y be a random variable with conditional distribution $P(y|\ x)$ for a fixed x.

As stated by Vapnik [1992], the problem of learning is that of choosing from the given set of functions the one which approximates best the supervisor's response. A good approximation is usually defined with the help of a loss function. The loss function $L(\hat{y}, y)$ is used to measure the inconsistency between the predicted value $\hat{y} = h(x)$ and the actual label y. It is a non-negative value, where the robustness of the model increases along with the decrease of the value of the loss function.
Thus, the goal of supervised learning is to choose a hypothesis h that minimise the expected loss (also called risk function)

$$\hat{L}(h) = E_{x,y}[L(h(x), y)] = \int L(h(x), y) dP(x, y) \tag{10.1.1}$$

where $L(\cdot, y)$ is the loss function. That is,

$$h^* = \arg\min_h E_{x,y}[L(h(x), y)]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Hence, the network is a distributed representation of the mapping.

***Remark* 10.1.1** *This formulation implies that learning corresponds to the problem of function approximation. Thus, supervised learning can be seen as a function approximator.*

10.1.1.2 Empirical risk minimisation

In machine learning, given the samples $\{(x_i, y_i); i = 1, 2, ..., m\}$, of m pairs of input vector $x_i \in \mathbb{R}^{N_i}$ and labelled output vectors $y_i \in \mathbb{R}^{N_o}$, a task for supervised learning consists in learning a functional relation between the input x_i and the desired output y_i, for the training set. That is, $F^* : \mathbb{R}^{N_i} \to \mathbb{R}^{N_o}$ is a generic function and $\{\big(x_i, F^*(x_i)\big); i = 1, ..., m\}$ is the sample or training set. We can view supervised learning as a map from $\mathbb{R}^{N_i}$ to $\mathbb{R}^{N_o}$ where the information about the mapping is stored in a weight vector $w \in \mathcal{W}$. Given a map $F = h(\cdot, w) : \mathbb{R}^{N_i} \to \mathbb{R}^{N_o}$ and a loss function L, we train the network by adjusting the weights such that the resulting map is close to the desired map. That is,

$$w^* = \arg\min_{w \in \mathcal{W}} L\big(h(x_i, w), y_i\big)_{i=1}^m \tag{10.1.2}$$

Thus, the learning can be understood as finding a mapping $h(\cdot, w)$ such that $h(x_i, w) \approx y_i$ with w an N_w dimensional vector of parameters to be learned.
As discussed in the previous section, in a probabilistic model, the goal of supervised learning is to choose a hypothesis h that minimise the expected loss in Equation (10.1.1). Since h is a function of the weight vector, we rewrite the risk functional (RF) as

$$\widehat{L}(w) = \int L(h(x, w), y) dP(x, y)$$

Here, we want to minimise $\widehat{L}(w)$ over the class of functions $h(x, w)$, $w \in \mathcal{W}$. An optimal scenario should allow for the algorithm to correctly determine the class labels for unseen instances. However, in most cases we do not know the joint probability distribution $P(x, y)$ to solve the expectation, and the only available information is contained in the training set.
One solution is to consider the induction principle of Empirical Risk Minimisation (ERM) (see Vapnik [1992]):

- we substitute the sample mean for the expectation
- we find an hypothesis that minimise the empirical loss (empirical risk functional):

$$\overline{L}(w) = \frac{1}{m} \sum_{i=1}^m L(h(x_i, w), y_i)$$

  Also known as (A.K.A.) sample average approximation (see Appendix (24.3)).

We can also write the loss function as $L(w; x_i, y_i)$. In that setting, the training is done by solving

$$w_m^* = \arg\min_{w \in \mathcal{W}} \overline{L}(w)$$

It is the hard core of empirical risk function and a significant component of structural risk function.
Under certain assumptions about the sequence of random variables $\{(x_i, y_i)\}$ (they are generated by a finite Markov process), if the set of hypotheses being considered is small enough, the minimiser of the empirical risk will closely approximate the minimiser of the expected risk as $m \to \infty$. Thus, for large enough m, we assume that the empirical loss approximates the expected value of the loss, $\overline{L} \approx \widehat{L}$. That is, we assume that the function $h(x, w_m^*)$ minimising $\overline{L}(w)$ over the set $w \in \mathcal{W}$, results in a risk (expected loss) $\widehat{L}(w_m^*)$ which is close to its minimum. Note, several methods such as least square, or, maximum likelihood (see Section (11.2.2)), are realisation of the ERM principle.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

10.1.1.2.1 Convergence Formally, we need to assess if the risk functional $\widehat{L}(w_m^*)$ converges to its minimum value on the set $w \in \mathcal{W}$ when $m \to \infty$, and find out the speed of convergence as m increases. Vapnik et al. [1989] showed it was equivalent to finding whether the empirical risk $\overline{L}(w)$ was converging uniformly to the actual risk $\widehat{L}(w)$ over the full set $h(x, w)$ for $w \in \mathcal{W}$, and what was the rate of convergence. Vapnik [1982] showed that bounds on the rate of uniform convergence were based on a quantitative measure of the capacity of the set of functions implemented by the learning machine: the VC-dimension of the set (from Vapnik-Chervonenkis dimension).
As an example, he considered the case of binary pattern recognition, for which $y \in \{0, 1\}$ and $h(x, w)$ is the class of indicator functions so that the loss function takes only two values $L(h(x, w), y) = 0$ if $y = h$ and it is equal to 1 otherwise. As a result, the risk functional is the probability of error, $P(w)$, and the empirical function, denoted $\nu(w)$, is the frequency of error. The VC-dimension of a set of indicator functions is the maximum number κ of vectors which can be shattered in all possible 2^κ ways using functions in the set. Bounds were obtained:

$$P(w) < \nu(w) + C_0(\frac{m}{\kappa}, \nu)$$

with confidence interval

$$C_0(\frac{m}{\kappa}, \nu) = \sqrt{\frac{\kappa\left(\ln \frac{2m}{\kappa} + 1\right) - \ln \nu}{m}}$$

which depends on the ratio $\frac{m}{\kappa}$.

10.1.1.2.2 Structural risk minimisation Based on the above example, we get the following two cases:

1. When the ratio $\frac{m}{\kappa}$ is large, the confidence interval gets small, and the actual risk is then bounded by only the empirical risk ν.
2. However, if the ratio is small, the confidence interval cannot be neglected, and the minimisation of $P(w)$ requires a new principle, based on the simultaneous minimisation of $\nu(w)$ and the confidence interval.

In case (2), one must therefore control the VC-dimension of the learning machine. This is done by introducing a nested structure of subsets $S_p = \{h(x, w); w \in \mathcal{W}_p\}$ such that

$$S_1 \subset S_2 \subset \cdots \subset S_n$$

where corresponding VC-dimensions satisfy

$$\kappa_1 < \kappa_1 < \cdots < \kappa_n$$

The principle of structure risk minimization (SRM) requires: the empirical risk ν has to be minimised for each element of the structure. The optimal element S^* is then selected to minimise the guaranteed risk, defined as the sum of the empirical risk and the confidence interval. However, we must face a tradeoff:

- as κ increases the minimum empirical risk decreases, but
- the confidence interval increases. Thus, the generalisation error is proportional to the variance.

Thus, SRM uses the VC-dimension as a controlling parameter for minimisation of generalisation bound. When designing statistical learning from data models, we can choose one of the following two approaches:

1. choose the appropriate structure and, keeping the confidence interval fixed, minimise the training error (empirical risk function) (by increasing κ). This is the approach in neural networks.
2. keep the value of the training error fixed, and minimise the confidence interval (linked to the variance). This is the approach in the SVM (see Section (11.3)).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

10.1.1.3 Model selection

In supervised learning, neural networks (NNs) are assessed based on the accuracy of their predictive response when provided with unseen input data.

10.1.1.3.1 Bonds for neural networks The **total risk** of a neural network is made of

1. the approximation error (bias): the distance between the target function and the closest NN function of a given architecture, and
2. the estimation error (variance): the distance between this ideal network function and an estimated network function.

Some authors studied the overall accuracy of a network when estimating an unknown target function. The approximation error (1) can be made as small as desired by increasing the number of nodes (see Hornik et al. [1989]). White [1990] showed that the total risk of an estimated NN converge to zero as the sample size and number of nodes increases to infinity. However, a large number of nodes makes it difficult to accurately estimate the parameters of the network for moderate sample size. Haussler [1992] focussed on the estimation error (2). In the case of nonparametric statistical theory of curve estimation, it was shown that total risk could be dealt effectively when functions were restricted by general smoothness properties (see Hardle [1990]). In parametric family of curves, some authors studied the case where the dimension of the family is increased at a certain rate as a function of the sample size, getting the smallest possible total risk, uniformly over classes of smooth functions. Further, the dimension of the family was allowed to grow, not as a function of the sample size, but determined from the data in order to optimise the selection criterion of the model (see Vapnik [1982]).

Ignoring the approximation error, Barron [1989] provided bounds on total risk of NNs in terms of an index of resolvability expressed in terms of the approximation error, the complexity of the networks, and the sample size. Assuming target functions to satisfy certain smoothness property expressed with the Fourier transform, Barron [1993] developed bounds the approximation error in a feedforward network with one layer of sigmoid nodes. Combining results, Barron [1994] developed bounds for the total risk of NNs, demonstrating convergence.

Given the sample $\{(x_i, y_i); i = 1, 2, ..., m\}$, of m pairs of input vector $x_i \in \mathbb{R}^{N_i}$ and labelled output vectors $y_i \in \mathbb{R}^{N_o}$, Barron [1994] considered the one-layer network, $F_{2,n}$, with sigmoidal activation function, Φ, given in Equation (5.1.6), where n stresses the number of nodes. Note, the sigmoid function is scaled with the parameter τ (see Section (5.1.2)). The framework allows for the possibility that the response is subject to error, that is, $y_i = F(x_i) + \epsilon_i$ where $E[\epsilon_i|x_i] = 0$. We let C_F be the first absolute moment of the Fourier magnitude distribution of F. We measure the **accuracy** of that network to the target function F in terms of the $L_2(\mu, B)$ norm as

$$\|F - F_{2,n}\|^2 = \int_B |F - F_{2,n}|^2 \mu(dx)$$

for an arbitrary probability measure μ with support B contained in the cube $[-1, 1]^{N_i}$. We let $\widehat{F}_{2,n}$ be the network estimated from the data and define the following measures:

- The norm $\|F - \widehat{F}_{2,n}\|$ measures the ability of the network to generalise to new data drawn with distribution μ. This is called the generalisation error (or statistical risk).
- The empirical risk $\frac{1}{m}\sum_{i=1}^m \left(F(x_i) - F_{2,n}(x_i)\right)^2$ only measures the accuracy at the observed data points $\{x_i; i = 1, ..., m\}$.

The approximation error of the best network of size n is defined by $\|F - F_{2,n}\|$. In order to obtain a bound on the generalisation error, one has to bound the approximation error. This was demonstrated in the following theorem from Barron [1993]:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 10.1.1 *Given an arbitrary sigmoidal function Φ, an arbitrary target function F with C_F finite, and a probability measure μ on a domain in $[-1,1]^{N_i}$, then for every $n \geqslant 1$, there exists an artificial neural network of the form in Equation (5.1.6) such that*

$$\|F - F_{2,n}\| \leqslant \frac{C_F}{\sqrt{n}}$$

The author incorporated the approximation bound into the analysis of the actual risk $\|F - \widehat{F}_{n,m}\|$ for the network $\widehat{F}_{n,m}$ minimising a penalised version of the above empirical risk. We consider a finite set $\Theta_{n,\epsilon,\tau,C}$ of parameter vectors θ and let $L_{n,m}(\theta)$ be a non-negative number such that $e^{-L_{n,m}(\theta)}$ is a prior probability for $\theta \in \Theta_n$. For each n, m and C, we let $\widehat{F}_{n,m,C}(x)$ be the least square estimator with θ restricted to $\Theta_{n,\epsilon,\tau,C}$. We let $\widehat{F}_{n,m}$ be a neural network estimated by complexity regularisation (constructed with a complexity penalty) and $\widehat{F}_m(x) = \widehat{F}_{\hat{n},m}(x)$ be the minimum complexity regularisation estimator, where $\hat{n}$ is that network size.
For a common class of ANNs, the mean integrated squared error between the estimated network and the target function F is bounded by

$$O(\frac{C_F^2}{n}) + O(\frac{nN_i}{m}\log m)$$

Thus, with $n \sim \left(\frac{m}{N_i \log m}\right)^{\frac{1}{2}}$ nodes, the order of the optimised bound is $O\big((\frac{N_i}{m})\log m\big)^{\frac{1}{2}}$. Note,

- $O(\frac{C_F^2}{n})$ represents the bound for model complexity, and
- $O\big(\frac{nN_i}{m}\log m\big)$ represents the bound associated to variance.

More formally, we get the theorem:

Theorem 10.1.2 *Given the two network estimators $\widehat{F}_{n,m}$ and $\widehat{F}_m(x)$. Under certain conditions, the statistical risk satisfies*

$$E\|F - \widehat{F}_{n,m}\|^2 \leqslant O(\frac{C_F^2}{n}) + O(\frac{nN_i}{m}\log m)$$

and

$$E\|F - \widehat{F}_m\|^2 \leqslant O\Big((\frac{N_i}{m}\log m)^{\frac{1}{2}}\Big)$$

respectively.

10.1.1.3.2 Bias-variance tradeoff Considering the convergence issues with supervised learning discussed above, Geman et al. [1992] presented a tradeoff between bias and variance, called the Bias-Variance tradeoff. It is the property of a set of predictive models $\mathcal{M}$, whereby models with a lower bias in parameter estimation have a higher variance of the parameter estimates across samples, and vice versa. The Bias-Variance dilemma is the conflict in trying to simultaneously minimise these two sources of error that prevent supervised learning algorithms from generalising beyond their training set. Three sources contribute to the prediction error summarised as follows:

- Bias: Imagine that we have available several different, but equally good, training data sets $\{(x_{i,k}, y_{i,k}); i = 1, 2, ..., m; k = 1, ..., K\}$, of m pairs of input vector $x_{i,k} \in \mathbb{R}^{N_i}$ and labelled output vectors $y_{i,k} \in \mathbb{R}^{N_o}$. A learning algorithm is biased for a particular input x if, when trained on each of these data sets, it is systematically incorrect when predicting the correct output for x.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Variance: A learning algorithm has high variance for a particular input x if it predicts different output values when trained on different training sets. It is the sensitivity of a model to changes to the training dataset.
- Irreducible error: Is is due to the noise in the problem itself and cannot be reduced regardless of the algorithm chosen.

This is illustrated in the bulls-eye diagram in Figure (10.1)

Figure 10.1: Bias and variance diagram.

When both the bias and variance are low, the difference between expected and predicted values does not differ significantly. When both the bias and the variance are high, that difference vary widely. The prediction error of a learned classifier is related to the sum of the bias and the variance of the learning algorithm. It is not possible to simultaneously minimise these two sources of error, preventing generalisation in supervised learning beyond the training datasets. More formally, given the sample $\{(x_i, y_i); i = 1, 2, ..., m\}$, of m pairs of input vector $x_i \in \mathbb{R}^{N_i}$ and labelled output vectors $y_i \in \mathbb{R}^{N_o}$, we assume that there is a function with noise (see Barron [1993])

$$y = h(x) + \epsilon$$

where the noise ϵ has zero mean and variance σ^2. We want to find the function $\widehat{h}(x)$ approximating $h(x)$ as well as possible by measuring the mean squared error between y and $\widehat{h}(x)$. That is, minimise the distance $\left(y - \widehat{h}(x)\right)$ both for the sample and for points outside that sample. As discussed above, whichever network we choose, we can decompose the expected error as follows:

$$E[\left(y - \widehat{h}(x)\right)^2] = \left(Bias(\widehat{h}(x))\right)^2 + Var(\widehat{h}(x)) + \sigma^2$$

where

$$Bias(\widehat{h}(x)) = E[\widehat{h}(x)] - h(x) \text{ and } Var(\widehat{h}(x)) = E[(\widehat{h}(x))^2] - (E[\widehat{h}(x)])^2$$

and σ^2 is the irreducible error Note, the expectation ranges over different choices of the training set $\{(x_i, y_i); i = 1, 2, ..., m\}$, all sampled from the same joint distribution $P(x, y)$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

A learning algorithm must be flexible enough (low bias) to fit the data, but not too flexible to avoid high variance. As discussed above, in some cases this tradeoff can be controlled by the capacity of the model. Some algorithms provide bias / variance parameter to adjust for this tradeoff. This is the case when adding a regularisation term to the loss function (see Equation (10.1.5)) (see SVM in Section (11.3)).

10.1.1.3.3 Overfitting In statistics, overfitting and underfitting are defined as follows:

- Overfitting is the production of an analysis that corresponds too closely, or exactly, to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably. An overfitted model is a statistical model that contains more parameters than can be justified by the data. The essence of overfitting is to have unknowingly extracted some of the residual variation (such as the noise) as if that variation represented underlying model structure.
- Underfitting occurs when a statistical model cannot adequately capture the underlying structure of the data. An underfitted model is a model where some parameters or terms that would appear in a correctly specified model are missing.

In machine learning, the possibility of overfitting exists because the criterion used for selecting the model is not the same as the criterion used to judge the suitability of a model. For example, a model might be selected by maximising its performance on some set of training data, and yet its suitability might be determined by its ability to perform well on unseen data. Then, overfitting occurs when a model begins to memorise training data rather than learning to generalise from a trend. Note, it results in higher variance in the testing data. This is illustrated in Figure (10.2).

Figure 10.2: Bias and variance diagram.

To lessen the chance of, or amount of, overfitting, several techniques are available such as

- Model comparison
- Cross-validation
- Regularisation

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Early stopping: Too many iteration may lead to too much variance in the training data. Thus, once validation accuracy stop increasing, we can stop the iteration.
- Pruning
- Bayesian priors
- Dropout: Randomly drop out some nodes of the network.

The basis of these techniques is either (1) to explicitly penalise overly complex models, or, (2) to test the model's ability to generalise by evaluating its performance on a set of data not used for training, which is assumed to approximate the typical unseen data that a model will encounter.

10.1.2 Solving the optimisation problem

10.1.2.1 The hypothesis function

In general, the relation between the input x_i and the desired output y_i can either be solved by a linear model as

$$\widehat{y}_i = W^\top x_i$$

where $W \in \mathbb{R}^{N_i \times N_o}$ is a weight matrix, or by a nonlinear model

$$\widehat{y}_i = f(W^\top x_i)$$

where $f(\cdot)$ is the activation function. There exists different ways of defining the activation function. Clevert et al. [2015] discussed how to find the most efficient ones in the case of deep learning.
As discussed in Section (5.1.3.2), a neural network is a circuit composed of neurons representing a map between multiple inputs and a single output. As such it is a parallel computation device. To simplify exposition, we consider a simple two-layer network with input layer X, weights W and output Y. For each neuron Y_i, the behaviour of the output neuron is given by the transition probability

$$P(Y_i = y|\, W_i, X_i) = f_i(y; W_i, X_i) \tag{10.1.3}$$

where y is a scalar and f_i is the conditional probability mass function (see Definition (24.1.17)) or activation function. Note, as discussed in Remark (24.6.1), the probability is the conditional probability $P(Y_i = y|W_i = w, X_i = x)$. It is the probability for a neuron Y_i to respond by y when given a certain input with certain synaptic weights. The same type of equation can be given for multilayer network, but in that case X is the output of a previous layer.
Some examples of neurons appearing in neural network models are:

- Linear non-linear (LN) neurons: These neurons first perform a linear action on their inputs, given by $h_i = W_i \cdot X_i$, and then a nonlinear function is applied to that value,

  $$P(Y_i = y|\, X_i, W_i) = f(y, h_i)$$

  Note, the output is random. In general, the activation function f is a sigmoidal function (see Section (5.1.2)), h_i determine the probability of getting a certain output. Some other examples of activation functions are:

  - Exponential linear unit (ELU): The function is

    $$f(x) = \begin{cases} x \text{ if } x \geqslant 0 \\ \alpha(e^x - 1) \text{ otherwise} \end{cases}$$

    where α is a parameter (see Clevert et al. [2015]).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Scaled exponential linear unit (SELU): It uses external normalisation techniques (like batch norm). The function is

$$f(x) = \lambda \begin{cases} x \text{ if } x \geqslant 0 \\ \alpha(e^x - 1) \text{ otherwise} \end{cases}$$

where α and λ are a parameters. Note, a custom weight initialisation technique is used with zero mean and variance $\frac{1}{N_i}$ where N_i is the input size.

- Rectified linear unit (ReLU): The function is

$$f(x) = \max(x, 0)$$

It removes negative values from an activation map by setting them to zero. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the layer (see Krizhevsky et al. [2017]). The gradient is zero in the region where x is negative, and the neuron is inactive. Otherwise it is one. Thus, there is no gradient close to zero, preventing vanishing gradients. Nonetheless, the values can be trapped in a dead state. LeCun et al. [2015] demonstrated the superiority of ReLU networks against the sigmoid or tanh in terms of their algorithmic learnability.

- Leaky ReLU: It is an alternative solution to the ReLU to avoid dead state (see He et al. [2015]). The function is

$$f(x) = \begin{cases} x \text{ if } x \geqslant 0 \\ \alpha x \text{ otherwise} \end{cases}$$

It permits a weak gradient, when the neuron is inactive (small α). It provides a self-imposed amount of risk for vanishing gradients.

- Softplus: It is a smooth version of the ReLU function given by

$$f(x) = \ln(e^x + 1)$$

- Bernoulli (binary) neurons: The neurons are binary, taking two values, typically 0 and 1. The probabilities are

$$P(Y_i = 1|X_i, W_i) = f(1, h_i) \text{ and } P(Y_i = 0|X_i, W_i) = f(0, h_i) = 1 - f(1, h_i)$$

We can equivalently say that it is the result of additive noise in the input. Denoting the noise by Z_i and $I(\cdot)$ the step function, we get

$$\begin{aligned} Y_i &= I(h_i + Z_i) \\ P(Y_i = +1) &= \int_{z > -h_i}^{\infty} P(z)dz \end{aligned}$$

- Logistic neurons: They are binary neurons that behave according to the logistic equation

$$P(Y_i = y) = f(h_i) = \frac{1}{1 + e^{-h_i}}$$

- Softmax: It is a special kind of activation function that produces a discrete probability distribution vector. Given a sample vector input X and weight vectors $\{W_i\}$, the predicted probability of $Y = j$ is

$$P(Y = j|\, X) = \frac{e^{X^\top W_j}}{\sum_{k=1}^{K} e^{X^\top W_k}} \tag{10.1.4}$$

The softmax function for output units is used to interpret the output values as posterior probabilities. In general, it is combined with a cross-entropy loss function.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

10.1.2.2 The loss function

There exists a large number of loss function (as defined in Equation (10.1.1)) applied to neural networks (see Janocha et al. [2017] in the case of classification tasks). Some examples of loss function are:

- The $0-1$ loss function: $L(\widehat{y}, y) = I(\widehat{y} \neq y)$, where $I(\cdot)$ is the indicator function.
- The mean square error (MSE) or quadratic loss function: $\frac{1}{m}\sum_{i=1}^{m}\big(y_i - h(x_i, w)\big)^2$ where $(y_i - \widehat{y}_i)$ is called the residual. It is widely used in linear regression. The method of minimising MSE is called ordinary least squares (OLS).
- L_2 norm: $\sum_{i=1}^{m}\big(y_i - h(x_i, w)\big)^2$. It is similar to the MSE but do not have the division by m.
- Other variants of the MSE exist such as the mean square logarithmic error where y_i is replaced by $\log(y_i + 1)$. Further, one can consider other norms than the L_2 norm such as the L_1 norm, $\sum_{i=1}^{m}\big|y_i - h(x_i, w)\big|$.
- Maximum likelihood estimation (MLE): Choose probabilities to maximise the likelihood of the observed data

$$L(w; x_i, y_i) = -\log P(Y = y_i|\, X = x_i)$$

  where $P(Y|X)$ is the transition probability. An example of transition probability is the softmax function

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

  where $s_{x_i} = h(x_i, w) = \widehat{y}_i$. In its simplest form, it can be used to measure the accuracy of a classifier or to estimate a density (see Bishop [1994]). It is given by

$$\overline{L}(w; x, y) = -\frac{1}{m}\sum_{i=1}^{m}\log \widehat{y}_i$$

- The Poisson loss function: It measures how the predicted distribution diverges from the expected one. It is a variant of the Poisson distribution which is widely used for modelling count data. It is the limitting distribution for a normal approximation to a binomial one where the number of trials goes to infinity and the probability goes to zero. It is given by

$$\overline{L}(w; x, y) = -\frac{1}{m}\sum_{i=1}^{m}\big(\widehat{y}_i - y_i \cdot \log \widehat{y}_i\big)$$

- Hinge loss function for SVMs: It is a convex approximation to the $0-1$ loss function. It is generally used for maximum-margin classification, most notably for support vector machines (SVMs). The function is given by

$$\overline{L}(w; x, y) = \sum_{j \neq y_i}\big(s_j - s_{y_i} + 1\big)^+$$

  where $s_{x_i} = h(x_i, w)$. In the special case of binary classification with intended output $y_i = \pm 1$ and classifier $\widehat{y}_i$ the hinge loss is given by

$$\overline{L}(w; x, y) = \frac{1}{m}\sum_{i=1}^{m}\big(1 - y_i \cdot \widehat{y}_i\big)^+$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

When y_i and $\widehat{y}_i$ have the same sign (meaning $\widehat{y}_i$ predicts the right class) and $|\widehat{y}_i| > 1$, the hinge loss equals to zero, but when they have opposite sign, it increases linearly with $\widehat{y}_i$ (one-sided error). A more general form is to replace 1 with the margin ξ_m as a customised value. The more you violate the margin, the higher the penalty is. Since it is a one-sided error, it is not well-suited for regression-based problems. A variant of the hinge function is to square the residual, getting the squared hinge loss function.

- ϵ-insensitive loss (proposed by Vapnik): It is identical in behaviour to the absolute loss function, except that any points within some selected range ϵ incur no error at all. It is given by

$$\overline{L}(w;x,y) = \frac{1}{m}\sum_{i=1}^{m} R(y_i,\widehat{y}_i)$$

where

$$R(y,\widehat{y}) = \begin{cases} 0 \text{ if } |y-\widehat{y}| \leqslant \epsilon \\ |y-\widehat{y}| - \epsilon \text{ otherwise} \end{cases}$$

This is an error-free margin function which is ideal for SVM. With a suitable selection of ϵ we obtain sparsity of solutions.

One can also consider measures of distance (see Section (2.2)). Some examples are

- Kullback-Leibler divergence: $D_{K,L}(P||Q) = \sum_y P(y)\log\frac{P(y)}{Q(y)}$ where P and Q are two distinct probability mass functions. The relative entropy (information divergence/gain) is a measure of how one probability distribution diverges from a second expected probability distribution. The KL loss function is given by

$$\begin{aligned} \overline{L}(w;x,y) &= \frac{1}{m}\sum_{i=1}^{m} D_{K,L}(y_i||\widehat{y}_i) \\ &= \frac{1}{m}\sum_{i=1}^{m}\big[y_i\cdot\log\frac{y_i}{\widehat{y}_i}\big] \\ &= \frac{1}{m}\sum_{i=1}^{m}\big(y_i\cdot\log y_i\big) - \frac{1}{m}\sum_{i=1}^{m}\big(y_i\cdot\log\widehat{y}_i\big) \end{aligned}$$

where the first term is the entropy and the second term is called the cross-entropy. A KL number 0 indicates that we expect similar distributions, while 1 indicates that the distribution are totally different. Since it is a distribution-wise asymmetric measure, it does not qualify as a statistical metric.

- Cross entropy: $H(P,Q) = H(P) + D_{K,L}(P||Q)$ where $H(\cdot)$ is the entropy (see Definition (2.2.1)). It can be used in binary classification. For instance, when the labels are assumed to take values 0 or 1, we have

$$R_{CE}(y,\widehat{y}) = \begin{cases} \log\widehat{y} \text{ if } y=1 \\ \log(1-\widehat{y}) \text{ if } y=0 \end{cases}$$

The loss function is given by

$$\overline{L}(w;x,y) = \frac{1}{m}\sum_{i=1}^{m}\big[y_i\cdot\log\widehat{y}_i + (1-y_i)\cdot\log(1-\widehat{y}_i)\big]$$

It measures the divergence between two probability distribution. When CE is large the difference between the two distributions is large, and when CE is small the two distributions are similar.
In the case where the labels are assumed to take values ± 1, we get

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$R_{CE}(y,\widehat{y}) = \begin{cases} -\log \widehat{y} \text{ if } y = 1 \\ -\log(1-\widehat{y}) \text{ if } y = -1 \end{cases}$$

In the case where we have multi-class, we define

$$\widehat{y}_t = \begin{cases} \widehat{y} \text{ if } y = 1 \\ 1-\widehat{y} \text{ if } y = -1 \end{cases}$$

and we get

$$R_{CE}(y,\widehat{y}) = R_{CE}(\widehat{y}) = -\log \widehat{y}_t$$

- Focal loss (FL): The loss function is reshaped to down-weight easy examples and thus focus training on hard negatives. In multi-class, it is defined as

$$R_{FL}(\widehat{y}_t) = -\big(1-\widehat{y}_t\big)^{\gamma} \log \widehat{y}_t$$

The modulating factor $\big(1-\widehat{y}_t\big)^{\gamma}$, with focusing parameter $\gamma > 0$, is added to the cross entropy loss. The parameter γ smoothly adjusts the rate at which easy examples are down-weighted. When $\gamma = 0$, R_{FL} is equivalent to R_{CE}. Likewise, when γ is increased, the effect of the modulating factor is increased.
One improvement is the α-balanced variant of FL defined as

$$R_{FL}(\widehat{y}_t) = -\alpha_t\big(1-\widehat{y}_t\big)^{\gamma} \log \widehat{y}_t$$

where α offset class imbalance.

10.1.2.3 Regularisation and stability

For the minimisation problem to have well-defined solution, we can place constraints on the set $\mathcal{H}$ of hypothesis being considered. One solution, discussed in Section (18.6.1), is to add a regularisation term to the loss function to avoid over-fitting on the training set (see bias-variance tradeoff above). We get

$$\overline{L}(w) = \frac{1}{m}\sum_{i=1}^{m} L(w; x_i, y_i)) + \lambda \cdot \Psi(w)$$

where λ is a regularisation parameter and $\Psi(\cdot)$ is a measure of distance. The main idea is to penalise large weights as they tend to result in overfitting. Some popular examples are:

- L_2 regularisation: $\Psi(w) = \|w\|_2^2 = w^{\top}w = \sum_{k,l} w_{k,l}^2$
- L_1 regularisation: $\Psi(w) = \|w\|_1 = \sum_{k,l} |w_{k,l}|$ which enforce sparsity of the weights.
- elastic net regularisation: $\Psi(w) = \sum_{k,l} \beta w_{k,l}^2 + |w_{k,l}|$

These approaches are part of the Tikhonov regularisation (see Section (18.6.1)). Some more complex regularisation methods are: Early stopping, Dropout, Batch normalisation, Stochastic depth, Fractional pooling, among others.

10.1.2.3.1 Early stopping While the error on the training set tends to decrease with the number of iterations, the error on the validation set usually starts to rise again once the network starts to overfit the training set. To avoid overfitting, we can stop training as soon as the error on the validation set reaches a minimum, that is, before the error on the validation set rises again. To do so, we check the performance on the validation set and save the model if a new minimum error is reached. When the error on the validation set begins to stall, training is stopped.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

10.1.2.3.2 Batch normalisation Batch normalisation is a technique for improving the speed, performance, and stability of artificial neural networks (see Ioffe et al. [2015]). We can make the networks robust to bad initialisation of weights by using batch normalisation. It alters the topology of the network by inserting normalisation operations between linearities and non-linearities of each dense layer. It is achieved through a normalisation step that fixes the means and variances of each layer's inputs. We let $\mathcal{B} = \{x_i; i = 1, ..., m\}$ be the mini-batch of the training set and consider the following statistics

- mean $\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$
- variance $\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}\left(x_i - \mu_{\mathcal{B}}\right)^2$
- normalise $\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$ where $\epsilon << 1$ is added for numerical stability. The resulting normalised activation has zero mean and near unit variance.
- scale and shift $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$. It restores the representation power of the network. The parameters γ, β are learnt in the optimisation process.

The batch normalisation applied to x over a mini-batch is described in Algorithm (21).

Algorithm 21 Batch normalisation

Require: mini-batch $\mathcal{B}$, parameters to be learned γ, β.
1: **for** $k = 1$ **to** m **do**
2: $\quad y_i \leftarrow \gamma\widehat{x}_i + \beta$
3: **end for**
4: **return** $\{y_i\}$

Batch normalisation reduces covariance shift by normalising and scaling inputs. The scale and shift parameters are trainable to avoid losing stability of the network.

10.1.2.3.3 Dropout Dropout is a regularisation technique patented by Google for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks (see Hinton et al. [2012], Srivastava et al. [2014]). The technique consists in dropping out some neurons between the training and testing period. During the training period we randomly ignore some activations with probability p. Then, during the testing period we use all activations but scale them by p. This approach efficiently prevent overfitting by reducing correlation between neurons. This is illustrated in Figure (10.3).

10.1.2.4 Solving for the optimum

Mathematically, the structural risk function of a model (as defined in Section (10.1.1)) can be represented as

$$w^* = \arg\min_{w \in \mathcal{W}} \overline{L}(w) + \lambda \cdot \Psi(w) \tag{10.1.5}$$

where $\Psi(\cdot)$ is a regularisation function. One solution is to perform the gradient descent (see Section (18.3.1.1)) and compute the gradient $\nabla_w \overline{L}(w)$, getting

$$\nabla_w \overline{L}(w) = \frac{1}{m}\sum_{i=1}^{m} \nabla_w L(w; x_i, y_i)) + \lambda \nabla_w \Psi(w)$$

However, when m is large it can be very expensive and one would rather consider the stochastic gradient descent (SGD), where the sum is approximated with mini-batch of examples. See details below.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 10.3: NN before and after dropout.

10.1.2.4.1 The batch gradient descent We can train the network by using batch gradient descent: all input values are processed and the weights are updated based on the average gradient. We minimise the empirical loss $\overline{L}$, assuming it is convex and unconstrained. The gradient descent on the empirical loss is described in Algorithm (22).

Algorithm 22 Gradient Descent on the Empirical Loss

1: **repeat**
2: $\quad w_{k+1} \leftarrow w_k - \eta_t \big(\frac{1}{m} \sum_{i=1}^{m} \frac{\partial L(w;x_i,y_i)}{\partial w} \big)$
3: **until** convergence or $k = K$

At each step, the gradient is the average of the gradient for all samples $i = 1, ..., m$. Thus, it is very slow when m is large.
Note, we can select (randomly) a subset $M \subseteq \{1, ..., m\}$, or mini-batch, of the training set and compute the gradient descent with updating equation

$$w_{k+1} \leftarrow w_k - \eta_t \big(\frac{1}{N_M} \sum_{i=1}^{N_M} \frac{\partial L(w; x_i, y_i)}{\partial w} \big)$$

where N_M is the number of sample in the mini-batch.

10.1.2.4.2 The stochastic gradient descent Alternatively, we can compute the stochastic gradient descent (SGD) from just one sample (or a few samples). The algorithm is described in Algorithm (23).

Algorithm 23 Stochastic Gradient Descent on the Empirical Loss

1: **repeat**
2: $\quad w_{k+1} \leftarrow w_k - \eta_t \frac{\partial L(w;x_i,y_i)}{\partial w}$
3: **until** convergence or $k = K$

Choose randomly one sample i and compute the gradient for that sample only. This approach reduces the number of data points used to compute the gradient and prevents overfitting to singular data points. At last, we bypass local

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

minima through random sampling. However, the gradient of one random sample is not the gradient of the objective function. Nonetheless, the SGD converges to the empirical loss minimum as well as to the expected loss minimum. But, convergence to the expected loss is slow

$$f(w_t) - E[f(w^*)] \leqslant O(\frac{1}{t}) \text{ or } O(\frac{1}{\sqrt{t}})$$

In principle, if the training set is small we consider the batch learning using quasi-Newton or conjugate gradient descent. On the other hand, if the training set is large we consider the stochastic gradient descent. However, the convergence is very sensitive to the learning rate. It needs to be determined by trial and error (model selection or cross-validation).

10.1.2.5 Online convex optimisation

Using the methods described in Section (18.3.2), Zinkevich [2003] proposed an algorithm for online convex optimisation. We let the feasible set $\chi \subseteq \mathbb{R}^n$ be a convex set. We have T convex cost functions $(c_1, c_2, ..., c_T)$, where each of the function is defined as $c_i : \chi \to [0, 1]$. See Algorithm (24).

Algorithm 24 Online Convex Algorithm

Require: : Import input vectors and corresponding labels.
1: Pick $x_1 \in \chi$ arbitrarily.
2: **for** $t = 1$ to T **do**
3: Step forward on the gradient direction: $\widehat{x}_{t+1} = x_t - \eta_t \nabla c_t(x_t)$
4: Project $\widehat{w}_{t+1}$ back to the feasible set: $x_{t+1} = Proj_\chi(\widehat{x}_{t+1})$
5: **end for**
6: **Return x**

Using the step size $\eta_t = \frac{1}{\sqrt{t}}$, the author proved that the regret of this online algorithm was bounded by

$$\sum_{t=1}^{T} c_t(x_t) - c_t(z_t) \leqslant \frac{D^2}{2}\sqrt{T} + G^2\sqrt{T} + 2D \cdot L(z_1, z_2, ..., z_T)\sqrt{T}$$

where $D = \max_{x,y\in\chi} \|x - y\|_2$ is the radius of the set, $\forall t, \forall x \in \chi$, $\|\nabla c_t(x)\|_2 \leqslant G$ is the upper bound of the gradient, and L is the total length of the drift, from z_1 to z_T given by $L(z_1, z_2, ..., z_T) = \sum_{i=1}^{T-1} \|z_{i+1} - z_i\|_2$.
As an example we consider the Hedge algorithm with n experts and construct a dimension for each expert, so that the feasible region lies in $\mathbb{R}^n$. Thus, we have

$$\chi = \{x : x_i \in [0, 1]; \sum_{i=1}^{n} x_i = 1\}$$

Then, a feasible vector x encodes a distribution over experts, where exactly one expert is chosen, and expert i is chosen with probability x_i. We want to find the projection

$$Proj(y) = \arg\min_{x\in\chi} \|y - x\|_2$$

10.1.3 Methods for stochastic optimisation

In Section (10.1.2), we have presented the optimisation problem and introduced some numerical methods. We saw that when the loss function was stochastic, it could be composed of a sum of sub-functions evaluated at different sub-samples of data. In that case the optimisation is made more efficient by taking gradient steps with respect to

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

the individual sub-functions, and we get the stochastic gradient descent (SGD). The loss functions can also have other sources of noise than data sub-sampling, such as dropout regularisation. This led to the development of various efficient stochastic optimisation techniques. For instance, Duchi et al. [2011] proposed the AdaGrad which works well with sparse gradients, and Tieleman et al. [2012] proposed RMSProp which works well in on-line and non-stationary settings. Other approaches include vSGD by Schaul et al. [2012], the AdaDelta by Zeiler [2012], among others. Kingma et al. [2015] introduced Adam, which is an algorithm for first-order gradient-based optimisation of stochastic objective functions, based on adaptive estimates of lower-order moments. Some advantages of this method are that the magnitudes of the parameter updates are invariant to rescaling of the gradient, the step sizes are approximately bounded by the step size hyperparameter, the loss function can be non-stationary, it works with sparse gradients.

10.1.3.1 Settings

We let $L(w)$ be a differentiable stochastic objective (or loss) function with parameters w, and we want to minimise its expected value $E[L(w)]$ with respect to its parameters w. We let $L_1(w), ..., L_T(w)$ be the realisation of that function at subsequent time steps $1, ..., T$. Further, we let $g_t = \nabla_w L_t(w)$ denote its gradient with respect to w. We let g_t^2 indicates the elementwise square $g_t \circ g_t$. All operations on vectors are elementwise.

10.1.3.2 The Adam algorithm

The Adam algorithm (see Kingma et al. [2015]) updates exponential moving averages of the gradient, denoted m_t, and the squared gradient, denoted v_t, where the parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages. Note, the moving averages are estimates on the first moment (mean) and the second raw moment (uncentered variance) of the gradient. Further, these moving averages are initialised as vector of zeros, which leads to biased moment estimates towards zero. This is strongest during the initial time step when the decay rates are small (the βs are close to 1). Fortunately, this initialisation bias can be counteracted, resulting in bias-corrected estimates $\widehat{m}_t$ and $\widehat{v}_t$.

We are now going to present the Adam algorithm. We let β_1^t and β_2^t denote β_1 and β_2 to the power t. It is displayed in Algorithm (25).

Algorithm 25 Adam Algorithm

Require: α: step size.
Require: $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for the moment estimates.
Require: $L(w)$: stochastic objective function with parameters w.
Require: w_0: initial parameter vector

```
m_0 ← 0 initialise first moment vector.
v_0 ← 0 initialise second moment vector.
t ← 0 initialise time step.
while w_t not converged do
    t ← t + 1
    g_t ← ∇_w L_t(w_{t-1}) get gradients with respect to objective at time step t.
    m_t ← β_1 · m_{t-1} + (1 − β_1) · g_t update biased first moment estimate.
    v_t ← β_2 · v_{t-1} + (1 − β_2) · g_t^2 update biased second raw moment estimate
    m̂_t ← m_t / (1 − β_1^t) compute bias-corrected first moment estimate.
    v̂_t ← v_t / (1 − β_2^t) compute bias-corrected second raw moment estimate.
    w_t ← w_{t-1} − α · m̂_t / (√v̂_t + ε)
end while
return w_t
```


This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The accuracy of that algorithm can be improved by changing the order of computation. That is, by replacing the last three lines in the loop with

$$\alpha_t = \alpha \cdot \frac{\sqrt{1-\beta_2^t}}{(1-\beta_1^t)} \text{ and } w_t \leftarrow w_{t-1} - \alpha_t \cdot \frac{m_t}{(\sqrt{v_t}+\widehat{\epsilon})}$$

10.1.3.3 The AdaGrad algorithm

The AdaGrad algorithm (see Duchi et al. [2011]) works well for sparse gradients. In its basic version, the parameters are updated as follows:

$$w_{t+1} = w_t - \alpha \cdot \frac{g_t}{\sqrt{\sum_{i=1}^t g_t^2}}$$

Kingma et al. [2015] noted that if we choose β_2 to be infinitesimally close to 1 from below, then $\lim_{\beta_2 \to 1} \widehat{v}_t = t^{-1} \cdot \sum_{i=1}^t g_t^2$. Thus, the AdaGrad corresponds to a version of Adam with $\beta_1 = 0$, infinitesimal $(1-\beta_2)$, and a replacement of α by an annealed version $\alpha_t = \alpha \cdot \alpha \cdot t^{-\frac{1}{2}}$, namely

$$\begin{aligned} w_t - \alpha \cdot t^{-\frac{1}{2}} \cdot \frac{\widehat{m}_t}{\lim_{\beta_2 \to 1} \widehat{v}_t} &= w_t - \alpha \cdot t^{-\frac{1}{2}} \cdot \frac{g_t}{\sqrt{t^{-1} \cdot \sum_{i=1}^t g_t^2}} \\ &= w_t - \alpha \cdot \frac{g_t}{\sqrt{\sum_{i=1}^t g_t^2}} \end{aligned}$$

Note, that this relation does not hold if we remove the bias-correction terms. In that case, a β_2 infinitesimally close to 1 would lead to infinitely large bias, and infinitely large parameter updates.

10.1.4 Solving the loss function

10.1.4.1 Data taking continuous values

We consider the samples $\{(x_i, y_i); i = 1, 2, ..., m\}$, of m pairs of input vector $x_i \in \mathbb{R}^{N_i}$ and labelled output vectors $y_i \in \mathbb{R}^{N_o}$. We want to learn the map $F = h(\cdot, w) : \mathbb{R}^{N_i} \to \mathbb{R}^{N_o}$ by minimising the loss function L in Equation (10.1.2). As discussed in Section (5.1), we train the network by adjusting the weights such that the resulting map is close to the desired map. The minimisation of the sum of the square error is given by

$$E(w) = \frac{1}{2} \sum_{i=1}^m \sum_{k=1}^{N_o} \big(h_k(x_i, w) - y_{i,k}\big)^2$$

where $y_{i,k}$ (or just y_k) is the kth element of the vector y_i, and $h_k(x, w)$ is the corresponding outputs of the network mapping function parametrised by w. In the limit of an infinite data set both bias and variance can be reduced to zero, and the optimal least squares solution is reached (see Bishop [1994]). In this limit, we can replace the infinite sum over data points in the above equation by an integral over the joint probability density. That is,

$$\begin{aligned} E(w) &= \lim_{m \to \infty} \frac{1}{2m} \sum_{i=1}^m \sum_{k=1}^{N_o} \big(h_k(x_i, w) - y_{i,k}\big)^2 \\ &= \frac{1}{2} \sum_{k=1}^{N_o} \int \int \big(h_k(x, w) - y_k\big)^2 p(y, x) dy dx \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where the factor $\frac{1}{m}$ has been introduced for convenience. In the case where the network h is very flexible, we can formally minimise the error function by functional differentiation with respect to $h_k(x, w)$:

$$\frac{\delta E}{\delta h_k(x, w)} = 0$$

Replacing in the error function, we get the minimisation function

$$h_k(x, w^*) =< y_k|\, x >$$

where w^* is the corresponding set of weight values. Note, we have used the conditional average of a quantity $Q(y)$ given by

$$< Q|\, x >= \int Q(y)p(y|x)dy$$

Hence, the network is given by the conditional average of the target data, conditioned on the input vector. This is illustrated in Figure (10.4).

Figure 10.4: Conditional average where x is the input variable and t is the output variable.

Rewriting the sum of squares errors in a different way, Bishop [1994] showed that the residual value of the error function at its global minimum was the average variance of the target data around its conditional average value. As a result, when training a standard neural network by least squares, we approximate two statistics of the target data: the conditional average value and the conditional average variance. If we know these two statistics, we can represent the conditional distribution of the target data by a Gaussian function with centre given by $h_k(x, w^*)$ and global variance parameter determined by the residual error.

10.1.4.2 Gaussian distribution

Assuming that the conditional distribution of the target data was Gaussian, Bishop [1994] obtained the least-squares formalism by using maximum likelihood. In that setting, the Gaussian conditional probability function is given by

$$p(y_k|x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(F_k(x)-y_k)^2}{2\sigma^2}}$$

where $F_k(x)$ is the mean of the target variable y_k. The conditional density of the complete target vector is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$p(y|x) = \prod_{k=1}^{N_o} p(y_k|x) = \frac{1}{(2\pi)^{N_o/2}\sigma^{N_o}} e^{-\frac{\sum_{k=1}^{N_o}(F_k(x)-y_k)^2}{2\sigma^2}}$$

Note, the function $F_k(x)$ is unknown and must be estimated. It is modelled by a parametric functional form $h_k(x, w)$, where the parameters w are determined from the training data $\{x_i, y_i\}$. Assuming the data is i.i.d., the likelihood of the data set is given by the product of the likelihoods for each of the data points:

$$L(w) = \prod_{i=1}^{m} p(y_i, x_i) = \prod_{i=1}^{m} p(y_i|\, x_i)p(x_i)$$

We can determine appropriate values for w by maximising the loss L. In practice, we minimise the negative logarithm of L, getting the error function

$$E = -\ln L$$

The author expressed the error function in the form

$$E = mN_o \ln\sigma + \frac{mN_o}{2}\ln 2\pi + \frac{1}{2\sigma^2}\sum_{i=1}^{m}\sum_{k=1}^{N_o}\big(h_k(x_i, w) - y_{i,k}\big)^2 + \sum_{i=1}^{m}\ln p(x_i)$$

Only the third term depends on the parameters w, so that we only need to minimise that term. Omitting the scaling factor $\frac{1}{\sigma^2}$, we recover the standard sum-of-squares error function used to train neural networks. We can therefore use standard techniques to find the optimum parameters w^*. The optimum value for σ can be found by minimising the error function E with respect to σ.

10.2 Dynamical systems

We have presented recurrent neural networks in Chapter (6), and will will discuss dynamical systems in Chapter (12). In this section we briefly present the mathematical formulation of such systems in supervised learning.

10.2.1 Mathematical formulation

We let the time-varying input signal be an N_ith order column vector $U(t) = [u_i(t)]$, the generated output is an N_oth order column vector $Y(t) = [y_o(t)]$ and the target output $\widehat{Y}(t)$ is an N_oth order column vector, where $t = 1, .., T$ and T is the number of data points in the training set $\{(U(t), \widehat{Y}_t)\}$. As discussed in Section (10.1.1), the goal of supervised learning is to learn a function

$$Y(t) = f(\cdots, U(t-1), U(t))$$

such that $E(Y, \widehat{Y})$ is minimised, where E is an error measure.
The relation between the input $U(t)$ and the desired output $\widehat{Y}(t)$ can either be solved by a linear model as

$$Y(t) = W^\top U(t)$$

where $W \in \mathbb{R}^{N_i \times N_o}$, or by a nonlinear model.
In nonlinear models, we generally expand nonlinearly the input $U(t)$ into a high dimensional feature vector $X(t) \in \mathbb{R}^{N_x}$, and use linear models to get a reasonable output vector $Y(t)$. The output vector can be written as

$$Y(t) = f_{out}(W_{out}^\top X(t)) = f_{out}(W_{out}^\top \phi(\cdots, U(t-1), U(t)))$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $f_{out}(\cdot)$ is the output function (identity, sigmoid, or other), and $W_{out} \in \mathbb{R}^{N_x \times N_o}$ are the trained output weights. The functions

$$X(t) = \phi(\cdots, U(t-1), U(t))$$

transforming the current input $U(t)$ and its history $U(t-1), \ldots$ into a higher dimensional vector $X(t)$ are called kernels, and we refer to them in machine learning as expansion methods. Since these kernels have an unbounded number of parameters, we define them recursively as

$$X(t) = \phi(X(t-1), U(t)) \tag{10.2.6}$$

Expansion methods includes, among others, Support Vector Machines, Feedforward Neural Networks, Radial Basis Function approximators, Slow Feature Analysis, various Probability Mixture models.
As an example of such kernels, the recurrent neural networks (RNNs) can be written as

$$X(t+1) = f\big(W_{res}^\top \cdot X(t) + W_{in}^\top \cdot U(t+1) + W_{fb}^\top \cdot Y(t)\big) \tag{10.2.7}$$

where $f(\cdot)$ is an activation function, $W_{in} \in \mathbb{R}^{N_i \times N_x}$ is an input weight matrix, $W_{res} \in \mathbb{R}^{N_x \times N_x}$ is a connection weight matrix, and $W_{fb} \in \mathbb{R}^{N_o \times N_x}$ is a feedback weight matrix.
In addition to the function $f(\cdot)$, leaky integrator neurons performs a leaky integration of its activation from previous time steps, which can be applied before or after the activation function $f(\cdot)$. In the latter case, assuming no feedback, we get

$$X(t+1) = (1 - a\Delta t)X(t) + \Delta t f\big(W_{in}^\top \cdot U(t+1) + W_{res}^\top \cdot X(t)\big)$$

where Δt is a compound time gap between two consecutive time steps divided by the time constant of the system, and a is the delay (or leakage) rate. Setting $a = 1$ and redefining Δt as a constant α controlling the speed of the dynamics, we get

$$X(t+1) = (1 - \alpha)X(t) + \alpha f\big(W_{in}^\top \cdot U(t+1) + W_{res}^\top \cdot X(t)\big) \tag{10.2.8}$$

which is an exponential moving average. In general, we set $a\Delta t \in [0, 1]$ in the first equation and $\alpha \in [0, 1]$ in the second such that a neuron neither retain, nor leak, more activation than it had. Thus, with one extra parameter we make sure that neuron activations $X(t)$ never go outside the boundary defined by $f(\cdot)$. In fact, the neuron activation performs a low-pass filtering of its activations with the cutoff frequency

$$f_c = \frac{a}{2\pi(1-a)\Delta t}$$

where Δt is the discretisation time step, allowing to tune the reservoirs for particular frequencies.
Non-temporal tasks using feedforward networks are functions, while RNNs may develop self-sustained temporal activation dynamics making them dynamical systems. Generally, supervised training of RNNs, such as gradient descent, adapt iteratively all weights according to their estimated gradients $\frac{\partial E}{\partial W}$ to minimise the output error $E = E(Y, \hat{Y})$. One can adapt backpropagation (BP) methods from feedforward neural networks in the case of RNNs, by propagating the gradient through network connections and time. For example, the Backpropagation Through Time (BPTT) has a runtime complexity of $O(N_x^2)$ per weight update per time step for a single output ($N_o = 1$). Alternatively, the Real-Time Recurrent Learning (RTRL) estimate the gradients recurrently, forward in time, but it has a runtime complexity of $O(N_x^4)$. Improvements in standard RNNs design were proposed independently by Maass et al. [2002] under the name of Liquid State Machines and Jaeger [2001] under the name of Echo State Networks. Over time, these types of models became known as Reservoir Computing.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

10.2.2 Handling memory

Maass et al. [2005a] [2005b] presented a computational theory characterising the gain in computational power that a fading memory system can acquire through feedback from trained readouts, in presence of noise. We briefly introduce this theory.
A map (or filter) F from input to output streams is defined to have fading memory if its current output at time t depends only on values of the input u during some finite time interval $[t-T,t]$.

Definition 10.2.1 *F has fading memory if there exists for every $\epsilon > 0$ some $\delta > 0$ and $T > 0$ so that*

$$|(Fu)(t) - (F\tilde{u})(t)| < \epsilon$$

for any $t \in \mathbb{R}$ and any input functions u, $\tilde{u}$ with $\|u(\tau) - \tilde{u}(\tau)\| < \delta$ for all $\tau \in [t-T,t]$.

This is a characteristic property of all filters that can be approximated by an integral over the input stream u, or more generally by Volterra or Wiener series.
In general, real-time computations involve memory or persistent internal states that can not be modelled with memory systems. It was shown that ANNs could be enlarged through feedback from trained readouts. As discussed above, ANNs are special cases of dynamical systems, which themselves can be seen as having universal capabilities for analog computing.

Theorem 10.2.1 *A large class $\mathcal{S}_n$ of systems of differential equations of the form*

$$x_i^{'}(t) = f_i\big(x_1(t),...,x_n(t)\big) + g_i\big(x_1(t),...,x_n(t)\big) \cdot v(t)\,,\, i = 1,...,n \tag{10.2.9}$$

where $v(t)$ is the input vector, are universal for analog computing in the following sense:
It can respond to an external input $u(t)$ with the dynamics of any n-th order differential equation of the form

$$z^{(n)}(t) = G\big(z(t), z^{'}(t), z^{''}(t),...,z^{(n-1)}(t)\big) + u(t) \tag{10.2.10}$$

for arbitrary smooth functions $G : \mathbb{R}^n \to \mathbb{R}$ if the input term $v(t)$ is replaced by a suitable memoryless feedback function $K\big(x_1(t),...,x_n(t),u(t)\big)$ and if a suitable memoryless readout function $h\big(x_1(t),...,x_n(t)\big)$ is applied to its internal state $\big(x_1(t),...,x_n(t)\big)$.
Also, the dynamic responses of all systems consisting of several higher order differential equations of the form Equation (10.2.10) can be simulated by fixed systems of the form Equation (10.2.9) with a corresponding number of feedbacks.

The authors characterised Equation (10.2.9) as follows:

$$x_i^{'}(t) = -\lambda_i x_i(t) + \sigma\big(\sum_{j=1}^{n} a_{ij} \cdot x_j(t)\big) + b_i \cdot v(t)\,,\, i = 1,...,n \tag{10.2.11}$$

where $\sigma(\cdot)$ is a standard activation function. Note, it is similar to Equation (10.2.8). Further, if the activation function σ is also applied to the term $v(t)$, the system in Equation (10.2.11) can still simulate arbitrary differential equations of the form Equation (10.2.10) with bounded inputs $u(t)$ and bounded responses $z(t),...,z^{(n-1)}(t)$.

As explained by Branicky [1995], all Turing machines can be simulated by systems of differential equations of the form Equation (10.2.10). Thus, systems of the form Equation (10.2.9), enlarged through feedback, are also universal for digital computing.
Casey [1996], Maass et al. [1998], among others, showed that additive noise, even with arbitrarily small bounded amplitude, reduces the non-fading memory capacity of any recurrent neural network to some finite number. As a result, arbitrary Turing machines can no-longer be simulated by such networks. Nonetheless, given this a priori limitation, feedback still provides noisy fading memory systems with maximum possible computational power. Thus, any finite state machine can be emulated by a fading memory system with feedback, despite the noise in the system.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 10.2.2 *Feedback allows linear and nonlinear fading memory systems, even in the presence of additive noise with bounded amplitude, to employ the computational capability and non-fading states of any given finite state machine for real-time processing of time varying inputs.*

Thus, the learning induced generation of high-dimensional attractors through feedback provides a new model for capturing the characteristics of persistent signal response.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 11

Statistical models

11.1 Introduction

11.1.1 Terminology

While statistics deal with data collection, organisation, analysis, interpretation and presentation, when applying statistics to a particular problem, we collect statistical population and devise statistical models. When census [1] data cannot be collected, statisticians collect data by developing specific experiment designs and survey samples. Sampling is the process of selecting a subset (a statistical sample) of individuals from within a statistical population to estimate characteristics of the whole population. Representative sampling assures that inferences and conclusions can reasonably extend from the sample to the population as a whole. A major problem lies in determining the extent that the sample chosen is actually representative. Statistics offers methods for estimating and correcting for any bias within the sample and data collection procedures (see Mann [1995]).
The statistical methods used in data analysis are:

1. Descriptive statistics: They summarise data from a sample using indexes such as the mean or standard deviation. That is, they are concerned with two sets of properties of a distribution (sample or population): central tendency (or location) seeks to characterise the distribution's central or typical value, while dispersion (or variability) characterises the extent to which members of the distribution depart from its center and each other.

2. Inferential statistics: They draw conclusions from data that are subject to random variation. It assumes that the data come from a larger population. Inferences on mathematical statistics are made under the framework of probability theory (see Appendix (24)), which deals with the analysis of random phenomena.

A standard statistical procedure involves a collection of data used to test the relationship between two statistical data sets, or, a data set and a synthetic data drawn from an idealised model. A hypothesis is proposed for the statistical relationship between the two data sets, and this is compared, as an alternative, to an idealised null hypothesis of no relationship between two data sets. Rejecting or disproving the null hypothesis is done using statistical tests that quantify the sense in which the null hypothesis can be proven false, given the data that are used in the test. Working from a null hypothesis, two basic forms of error are recognised:

- Type I errors (null hypothesis is falsely rejected giving a "false positive").
- Type II errors (null hypothesis fails to be rejected and an actual difference between populations is missed giving a "false negative").

[1] A census is the procedure of systematically acquiring and recording information about the members of a given population. The term is used mostly in connection with national population and housing censuses.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

When a census is not feasible, a chosen subset of the population called a sample is studied. Once a sample that is representative of the population is determined, data is collected for the sample members in an observational or experimental setting. Again, descriptive statistics can be used to summarise the sample data. However, the drawing of the sample has been subject to an element of randomness [2], hence the established numerical descriptors from the sample are also due to uncertainty. To still draw meaningful conclusions about the entire population, inferential statistics is needed. It uses patterns in the sample data to draw inferences about the population represented, accounting for randomness. More formally, statistical inference is the process of using data analysis to deduce properties of an underlying probability distribution. Any statistical inference requires some assumptions. A statistical model is a set of assumptions concerning the generation of the observed data and similar data. For instance, we consider a set of independent and identically distributed (i.i.d.) random variables with a given probability distribution. Standard statistical inference and estimation theory defines a random sample as the random vector given by the column vector of these i.i.d. variables. The population being examined is described by a probability distribution that may have unknown parameters. A statistic is a random variable that is a function of the random sample, but not a function of unknown parameters. However, the probability distribution of the statistic, may have unknown parameters. Considering a function of the unknown parameter, an estimator is a statistic used to estimate such function. An estimator is said to be unbiased if its expected value is equal to the true value of the unknown parameter being estimated, and asymptotically unbiased if its expected value converges at the limit to the true value of such parameter. Several methods have been proposed to obtain these estimators such as the method of moments, the maximum likelihood method, the least squares method and the method of estimating equations.

Any statistical inference requires some assumptions. A statistical model is a set of assumptions concerning the generation of the observed data and similar data. Descriptions of statistical models usually emphasize the role of population quantities of interest, about which we wish to draw inference. As such, descriptive statistics are used as a preliminary step before more formal inferences are drawn. When considering statistical models, there are three levels of modelling assumptions:

1. Fully parametric: The probability distributions describing the data-generation process are assumed to be fully described by a family of probability distributions involving only a finite number of unknown parameters.

2. Non-parametric: The assumptions made about the process generating the data are much less than in parametric statistics and may be minimal.

3. Semi-parametric: This term typically implies assumptions in between the fully and non-parametric approaches.

Correctly calibrated inference requires these assumptions to be correct, otherwise it will lead to invalidate statistical inference.

Inference can extend to forecasting, prediction and estimation of unobserved values either in or associated with the population being studied. It can include extrapolation and interpolation of time series or spatial data, and can also include data mining. A time series is a set of measurements recorded on a single unit over multiple time periods. More generally, a time series is a set of statistics, usually collected at regular intervals, and occurring naturally in many application areas such as economics, finance, environmental, medicine, etc. While the methods of time series analysis pre-date those for general stochastic processes and Markov Chains, their aims are to describe and summarise time series data, fit low-dimensional models, and make forecasts. Even though it is a highly empirical discipline, this theory forms the foundation for making inference.
In that setting, statistical models can be interpretable or predictive. Assuming a theory stating that input x causes response y via the function F, $y = F(x)$. Let x and y be measurable variables corresponding to sample taken from $\mathcal{X}$ and $\mathcal{Y}$, respectively, and let f me a model approximating F. Then,

[2] Randomness is the lack of pattern or predictability in events. A random sequence of events, symbols or steps has no order and does not follow an intelligible pattern or combination.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Interpretable modelling seeks f that best explains F.
- Predictive modelling seeks the function f that best predicts future y.

Ideally we want to both, get interpretability with predictive power. However, in general, there is a trade-off to be made between these two approaches.

11.1.2 Basic tools for summarising and forecasting data

We assume that we have available a database from which to filter data and build numerical forecasts, that is, a table with multiple dimensions. Cross sectional data refer to measurements on multiple units, recorded in a single time period. Although forecasting practice involves multiple series, the methods we are going to examine use data from the past and present to predict future outcomes. Hence, we will first focus on the use of time series data.

11.1.2.1 Presenting forecasting methods

Forecasting is about making statements on events whose actual outcomes have not yet been observed. We distinguish two main types of forecasting methods:

1. Qualitative forecasting techniques are subjective, based on opinions and judgements, and are appropriate when past data are not available. For example, one tries to verify whether there is some causal relationship between some variables and the demand. If this is the case, and if the variable is known in advance, it can be used to make a forecast.
2. On the contrary, quantitative forecasting models are used to forecast future data as a function of past data, and as such, are appropriate when past data are available. The main idea being that the evolution in the past will continue into the future. If we observe some correlations between some variables, then we can use these correlations to make some forecast. A dynamic model incorporating all the important internal and external variables is implemented and used to test different alternatives. For instance, to estimate the future demand accurately, we need to take into account facts influencing the demand.

Subjective forecasts are often time-consuming to generate and may be subject to a variety of conscious or unconscious biases. In general, simple analysis of available data can perform as well as judgement procedures, and are much quicker and less expensive to produce. The effective possible choices are judgement only, quantitative method only and quantitative method with results adjusted by user judgement. All three options have their place in the forecasting lexicon, depending upon costs, available data and the importance of the task in hand. Careful subjective adjustment of quantitative forecasts may often be the best combination, but we first need to develop an effective arsenal of quantitative methods. To do so, we need to distinguish between methods and models.

- A forecasting method is a (numerical) procedure for generating a forecast. When such methods are not based upon an underlying statistical model, they are termed heuristic.
- A statistical (forecasting) model is a statistical description of the data generating process from which a forecasting method may be derived. Forecasts are made by using a forecast function that is derived from the model.

For example, we can specify a forecasting method as

$$F_t = b_0 + b_1 t$$

where F_t is the forecast for time period t, b_0 is the intercept representing the value at time zero, and b_1 is the slope representing the increase in forecast values from one period to the next. All we need to do to obtain a forecast is

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

to calibrate the model. However, we lack a basis for choosing values for the parameters, and we can not assess the uncertainty inherent in the forecasts. Alternatively, we may formulate a forecasting model as

$$Y_t = \beta_0 + \beta_1 t + \epsilon$$

where Y denotes the time series being studied, β_0 and β_1 are the level and slope parameters, and ϵ denotes a random error term corresponding to that part of the series that cannot be fitted by the trend line. Once we make appropriate assumptions about the nature of the error term, we can estimate the unknown parameters, β_0 and β_1. These estimates are typically written as b_0 and b_1. Thus the forecasting model gives rise to the forecast function

$$F_t = b_0 + b_1 t$$

where the underlying model enables us to make statements about the uncertainty in the forecast, something that the heuristic method do not provide. As a result, risk and uncertainty are central to forecasting, as one must indicate the degree of uncertainty attaching to forecasts. Hence, some idea about its probability distribution is necessary. For example, assuming a forecast for some demand has the distribution of Gauss (normal) with average μ and standard deviation σ, the coefficient of variation of the prediction is $\frac{\sigma}{\mu}$.

11.1.2.2 Summarising the data

Following Brockwell et al. [1991], we write the real-valued series of observations as $..., Y_{-2}, Y_{-1}, Y_0, Y_1, Y_2, ...$ a doubly infinite sequence of real-valued random variables indexed by $\mathbb{Z}$. Given a set of n values $Y_1, Y_2, .., Y_n$, we place these values in ascending order written as $Y_{(1)} \leqslant Y_{(2)} \leqslant ... \leqslant Y_{(n)}$. The median is the middle observation. When n is odd it can be written $n = 2m + 1$ and the median is $Y_{(m+1)}$, and when n is even we get $n = 2m$ and the median is $\frac{1}{2}(Y_{(m)} + Y_{(m)})$. It is possible to have two very different datasets with the same means and medians. For that reason, measures of the middle are useful but limited. Another important attribute of a dataset is its dispersion or variability about its middle. The most useful measures of dispersion (or variability) are the range, the percentiles, the mean absolute deviation, and the standard deviation. The **range** denotes the difference between the largest and smallest values in the sample

$$Range = Y_{(n)} - Y_{(1)}$$

Therefore, the more spread out the data values are, the larger the range will be. However, if a few observations are relatively far from the middle but the rest are relatively close to the middle, the range can give a distorted measure of dispersion. **Percentiles** are positional measures for a dataset that enable one to determine the relative standing of a single measurement within the dataset. In particular, the pth percentile is defined to be a number such that $p\%$ of the observations are less than or equal to that number and $(100 - p)\%$ are greater than that number. So, for example, an observation that is at the 75th percentile is less than only 25% of the data. In practice, we often can not satisfy the definition exactly. However, the steps outlined below at least satisfies the spirit of the definition:

1. Order the data values from smallest to largest, including ties.
2. Determine the position
$$k.ddd = 1 + \frac{p(n-1)}{100}$$
3. The pth percentile is located between the kth and the $(k + 1)$th ordered value. Use the fractional part of the position, .ddd as an interpolation factor between these values. If $k = 0$, then take the smallest observation as the percentile and if $k = n$, then take the largest observation as the percentile.

The 50th percentile is the median and partitions the data into a lower half (below median) and upper half (above median). The 25th, 50th, 75th percentiles are referred to as quartiles. They partition the data into 4 groups with 25% of the values below the 25th percentile (lower quartile), 25% between the lower quartile and the median, $25th$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

between the median and the 75th percentile (upper quartile), and 25% bove the upper quartile. The difference between the upper and lower quartiles is referred to as the inter-quartile range. This is the range of the middle 50% of the data. Given $d_i = Y_i - \overline{Y}$ where $\overline{Y}$ is the arithmetic mean, the Mean Absolute Deviation (MAD) is the average of the deviations about the mean, ignoring the sign

$$MAD = \frac{1}{n}\sum_i |d_i|$$

The sample variance is an average of the squared deviations about the mean

$$S^2 = \frac{1}{n-1}\sum_i d_i^2$$

The population variance is given by

$$\sigma_p^2 = \frac{1}{n}\sum_i (Y_i - \mu_p)^2$$

where μ_p is the population mean. Note that the unit of measure for the variance is the square of the unit of measure for the data. For that reason (and others), the square root of the variance, called the standard deviation, is more commonly used as a measure of dispersion. Note that datasets in which the values tend to be far away from the middle have a large variance (and hence large standard deviation), and datasets in which the values cluster closely around the middle have small variance. Unfortunately, it is also the case that a dataset with one value very far from the middle and the rest very close to the middle also will have a large variance. Comparing the variance with the MAD, S gives greater weight to the more extreme observations by squaring them and it may be shown that $S > MAD$ whenever MAD is greater than zero. A rough relationship between the two is

$$S = 1.25 \cdot MAD$$

Chebychev's inequality (or Markov's inequality) guarantees that, for a wide class of probability distributions, no more than $\frac{1}{k^2}$ of the distribution's values can be more than k standard deviations away from the mean. Equivalently, at least $(1 - \frac{1}{k^2})$ of the distribution's values are within k standard deviations of the mean. That is, the standard deviation of a dataset can be interpreted by Chebychev's Theorem (see Tchebichef [1867]):

Theorem 11.1.1 *Chebychev's Theorem*
For any $k > 1$, the proportion of observations within the interval $\mu_p \pm k\sigma_p$ is at least $(1 - \frac{1}{k^2})$.

The inequality has great utility because it can be applied to any probability distribution in which the mean and variance are defined. Hence, knowing just the mean and standard deviation of a dataset allows us to obtain a rough picture of the distribution of the data values. Note that the smaller the standard deviation, the smaller is the interval that is guaranteed to contain at least 75% of the observations. Conversely, the larger the standard deviation, the more likely it is that an observation will not be close to the mean. Note that Chebychev's Theorem applies to all data and therefore must be conservative. In many situations the actual percentages contained within these intervals are much higher than the minimums specified by this theorem. If the shape of the data histogram is known, then better results can be given. In particular, if it is known that the data histogram is approximately bell-shaped (like the Gaussian distribution in Figure (11.1)), then we can say

- $\mu_p \pm \sigma_p$ contains approximately 68%,
- $\mu_p \pm 2\sigma_p$ contains approximately 95%,
- $\mu_p \pm 3\sigma_p$ contains essentially all

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

of the data values. This set of results is called the empirical rule. Several extensions of the Chebyshev's inequality have been developed, among which is the asymmetric two-sided version given by

$$P(k_1 < Y < k_2) \geqslant \frac{4\big((\mu_p - k_1)(k_2 - \mu_p) - \sigma_p^2\big)}{(k_2 - k_1)^2}$$

In mathematical statistics, a random variable Y is standardized by subtracting its expected value $E[Y]$ and dividing the difference by its standard deviation $\sigma(Y)$

$$Z = \frac{Y - E[Y]}{\sigma(Y)}$$

A Gaussian (or normal) distribution is a symmetric, continuous, bell-shaped distribution of a random variable fully characterised by its first two moments, a mean μ and a standard deviation σ, respectively. The probability density function (see Definition (24.1.9)) of the Gaussian law given by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

with mean μ and standard deviation σ. If Z is a standard normal deviate, then $X = \sigma Z + \mu$ will have a normal distribution with expected value μ and standard deviation σ. Conversely, if X is a normal deviate with parameters μ and σ^2, then the distribution $Z = \frac{X-\mu}{\sigma}$ will have a standard normal distribution. This variate is also called the standardised form of X. The cumulative distribution function (see Definition (24.1.15)) of the (standard) Gaussian law is given by

$$F(x) = P(X \leqslant x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2} du$$

For $x = 2$, we get $F(2) = 0.977$. We display in Figure (11.1) a normal probability distribution centered on mean $\mu = 0$.

Figure 11.1: Normal distribution curve.

About 68% of values drawn from a normal distribution are within one standard deviation σ. away from the mean. About 95% of the values lie within two standard deviations, and about 99.7% are within three standard deviations. More formally, the probability that a normal deviate lies in the range between μ
$pmk \cdot \sigma$ is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$p = F(\mu + k \cdot \sigma) - F(\mu - k \cdot \sigma) = erf(\frac{k}{\sqrt{2}})$$

where erf is the error function. We display in Table (11.1) the range of a normal deviate for $k = 1, 2, 3$.

k	p	1-p
1	0.682	0.317
2	0.954	0.045
3	0.997	0.002

Table 11.1: Range of a normal deviate

For large k, one can use the approximation

$$1 - p \approx \frac{e^{-\frac{1}{2}k^2}}{k\sqrt{\frac{\pi}{2}}}$$

For a normal random variable with mean μ and variance σ^2, the quantile function is

$$F^{-1}(p) = \mu + \sigma\sqrt{2}erf^{-1}(2p - 1)\ , p \in (0, 1)$$

The Z-score is a dimensionless quantity obtained by subtracting the population mean μ_p from an individual raw score Y_i and then dividing the difference by the population standard deviation σ_p. That is,

$$Z = \frac{Y_i - \mu_p}{\sigma_p}$$

From Chebychev's theorem (see Theorem (11.1.1)), at least 75% of observations in any dataset will have Z-scores in the range $[-2, 2]$. The standard score is the (signed) number of standard deviations an observation or datum is above the mean, and it provides an assessment of how off-target a process is operating. The use of the term Z is due to the fact that the Normal distribution is also known as the Z distribution. They are most frequently used to compare a sample to a standard normal deviate, though they can be defined without assumptions of normality. Note, considering the Z-score, Cantelli obtained sharpened bounds given by (see Grimmett et al. [1992])

$$P(Z \geqslant k) \leqslant \frac{1}{1 + k^2}$$

The Z-score is only defined if one knows the population parameters, but knowing the true standard deviation of a population is often unrealistic except in cases such as standardized testing, where the entire population is measured. If one only has a sample set, then the analogous computation with sample mean and sample standard deviation yields the Student's t-statistic. Given a sample mean $\overline{Y}$ and sample standard deviation S, we define the standardised scores for the observations, also known as Z-scores as

$$Z = \frac{Y_i - \overline{Y}}{S}$$

The Z-score is to examine forecast errors and one proceed in three steps:

- Check that the observed distribution of the errors is approximately normal
- If the assumption is satisfied, relate the Z-score to the normal tables
  - The probability that $|Z| > 1$ is about 0.32

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

  - The probability that $|Z| > 2$ is about 0.046
  - The probability that $|Z| > 3$ is about 0.0027

- Create a time series plot of the residuals (and/or Z-scores) when appropriate to determine which observations appear to be extreme

Hence, whenever you see a Z-score greater than 3 in absolute value, the observation is very atypical, and we refer to such observations as outliers.

The change in the absolute level of the series from one period to the next is called the first difference of the series, given by

$$DY_t = Y_t - Y_{t-1}$$

where Y_{t-1} is known at time t. Letting $\hat{D}_t$ be the forecast for the difference, the forecast for Y_t becomes

$$F_t = \hat{Y}_t = Y_{t-1} + \hat{D}_t$$

The growth rate for Y_t is

$$G_t = GY_t = 100\frac{D_t}{Y_{t-1}}$$

so that the forecast for Y_t can be written as

$$F_t = \hat{Y}_t = Y_{t-1}\big(1 + \frac{\hat{G}_t}{100}\big)$$

If we think of changes in the time series in absolute terms we should use DY and if we think of it in relative terms we should use GY. Note, reducing a chocolate ration by 50% and then increasing it by 50% does not gives you as much chocolate as before since

$$(1 - \frac{50}{100})(1 + \frac{50}{100}) = 0.75$$

To avoid this asymmetry we can use the logarithm transform $L_t = \ln Y_t$ with first difference in logarithm being

$$DL_t = \ln Y_t - \ln Y_{t-1}$$

converting the exponential (or proportional) growth into linear growth. If we generate a forecast of the log-difference, the forecast for the original series, given the previous value Y_{t-1} becomes

$$\hat{Y}_t = Y_{t-1}e^{D\hat{L}_t}$$

11.1.2.3 Measuring the forecasting accuracy

When selecting a forecasting procedure, a key question is how to measure performance. A natural approach would be to look at the differences between the observed values and the forecasts, and to use their average as a performance measure. Suppose that we start from forecast origin t so that the forecasts are made successively (one-step-ahead) at times $t+1, t+2, ..., t+h$, there being h such forecasts in all. The one-step-ahead forecast error at time $t+i$ may be denoted by

$$e_{t+i} = Y_{t+i} - F_{t+i}$$

The Mean Error (ME) is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$ME = \frac{1}{h}\sum_{i=1}^{h}(Y_{t+i} - F_{t+i}) = \frac{1}{h}\sum_{i=1}^{h} e_{t+i}$$

ME will be large and positive (negative) when the actual value is consistently greater (less) than the forecast. However, this measure do not reflect variability, as positive and negative errors could virtually cancel each other out, yet substantial forecasting errors could remain. Hence, we need measures that take account of the magnitude of an error regardless of the sign. The simplest way to gauge the variability in forecasting performance is to examine the absolute errors, defined as the value of the error ignoring its sign and expressed as

$$|e_i| = |Y_i - F_i|$$

We now present various averages, based upon the errors or the absolute errors.

- the Mean Absolute Error

$$MAE = \frac{1}{h}\sum_{i=1}^{h}|Y_{t+i} - F_{t+i}| = \frac{1}{h}\sum_{i=1}^{h}|e_{t+i}|$$

- the Mean Absolute Percentage Error

$$MAPE = \frac{100}{h}\sum_{i=1}^{h}\frac{|Y_{t+i} - F_{t+i}|}{Y_{t+i}} = \frac{100}{h}\sum_{i=1}^{h}\frac{|e_{t+i}|}{Y_{t+i}}$$

- the Mean Square Error

$$MSE = \frac{1}{h}\sum_{i=1}^{h}(Y_{t+i} - F_{t+i})^2 = \frac{1}{h}\sum_{i=1}^{h} e_{t+i}^2$$

- the Normalised Mean Square Error

$$NMSE = \frac{1}{\sigma^2 h}\sum_{i=1}^{h}(Y_{t+i} - F_{t+i})^2 = \frac{1}{\sigma^2 h}\sum_{i=1}^{h} e_{t+i}^2$$

  where σ^2 is the variance of the true sequence over the prediction period (validation set).

- the Root Mean Square Error

$$RMSE = \sqrt{MSE}$$

- the Mean Absolute Scaled Error

$$MASE = \frac{\sum_{i=1}^{h}|Y_{t+i} - F_{t+i}|}{\sum_{i=1}^{h}|Y_{t+i} - Y_{t+i-1}|}$$

- the Directional Symmetry

$$DS = \frac{1}{h-1}\sum_{i=1}^{h}\mathcal{H}(Y_{t+i}.F_{t+i})$$

  where $\mathcal{H}(x) = 1$ if $x > 0$ and $\mathcal{H}(x) = 0$ otherwise (Heaviside function).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- the Direction Variation Symmetry

$$DVS = \frac{1}{h-1}\sum_{i=2}^{h}\mathcal{H}((Y_{t+i}-Y_{t+i-1}).(F_{t+i}-F_{t+i-1}))$$

Note, the Mean Absolute Scaled Error was introduced by Hyndman et al. [2006]. It is the ratio of the MAE for the current set of forecasts relative to the MAE for forecasts made using the random walk. Hence, for $MASE > 1$ the random walk forecasts are superior, otherwise the method under consideration is superior to the random walk. We now give some general comments on these measures.

- MAPE should only be used when $Y > 0$, MASE is not so restricted.
- MAPE is the most commonly used error measure in practice, but it is sensitive to values of Y close to zero.
- MSE is measured in terms of (dollars), and taking the square root to obtain the RMSE restores the original units.
- A value of the $NMSE = 1$ corresponds to predicting the unconditional mean.
- The RMSE gives greater weight to large (absolute) errors. It is therefore sensitive to extreme errors.
- The measure using absolute values always equals or exceeds the absolute value of the measure based on the errors, so that $MAE \geqslant |ME|$ and $MAPE \geqslant |MPE|$. If the values are close in magnitude that suggests a systematic bias in the forecasts.
- Both MAPE and MASE are scale-free and so can be used to make comparisons across multiple series. The other measures need additional scaling.
- DS is the percentage of correctly predicted directions with respect to the target variable. It provides a measure of the number of times the sign of the target was correctly forecast.
- DVS is the percentage of correctly predicted direction variations with respect to the target variable.

11.1.2.4 Prediction intervals

So far, we have considered point forecasts which are future observations for which we report a single forecast value. However, confidence in a single number is often misplaced. We assume that the predictive distribution for the series Y follows the normal law (although such an assumption is at best an approximation and needs to be checked). If we assume that the standard deviation (SD) of the distribution is known, we may use the upper 95% point of the standard normal distribution in Figure (11.1) (this value is 1.645 since $F(1.645) = 0.95$) so that the one-sided prediction interval is

$$\hat{Y} + 1.645 \times SD$$

where $\hat{Y}$ is the point forecast. The normal distribution being the most widely used in the construction of prediction intervals, it is critical to check that the forecast errors are approximately normally distributed. Typically the SD is unknown and must be estimated from the sample that was used to generate the point forecast, meaning that we use the RMSE to estimate the SD (see Section (11.1.2.3)). We can also use the two-sided $100(1-\alpha)$ prediction intervals given by

$$\hat{Y} \pm z_{1-\frac{\alpha}{2}} \times RMSE$$

where $z_{1-\frac{\alpha}{2}}$ denotes the upper $100(1-\frac{\alpha}{2})$ percentage point of the normal distribution. In the case of a 95% one-step-ahead prediction intervals we set $\alpha = 5\%$ and get $z_{1-\frac{0.05}{2}} = 1.96$. The general purpose of such intervals is to provide an indication of the reliability of the point forecasts.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

An alternative approach to using theoretical formulae when calculating prediction intervals is to use the observed errors to show the range of variation expected in the forecasts. For instance, we can calculate the 1-step ahead errors made using the random walk forecasts which form a histogram. We can also fit a theoretical probability density to the observed errors. Other distributions (than normal) are possible since in many applications more extreme errors are observed than those suggested by a normal distribution. Fitting a distribution gives us more precise estimates of the prediction intervals which are called empirical prediction intervals. To be useful, these empirical prediction intervals need to be based on a large sample of errors.

11.1.2.5 Estimating model parameters

Usually we partition the original series into two parts and refer to the first part (containing $75-80\%$ of the observations) as the estimation sample, which is used to estimate the starting values and the smoothing parameters. The parameters are commonly estimated by minimizing the mean squared error (MSE), although the mean absolute error (MAE) or mean absolute percentage error (MAPE) are also used. The second part called hold-out sample represents the remaining $20-25\%$ of the observations and is used to check forecasting performance. Some programs allow repeated estimation and forecast error evaluation by advancing the estimation sample one observation at a time and repeating the error calculations.

When forecasting data, we should not rely on an arbitrary pre-set smoothing parameter. Most computer programs nowadays provide efficient estimates of the smoothing constant, based upon minimizing some measure of risk such as the mean squared error (MSE) for the one-step-ahead forecasts

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - F_i)^2$$

where $Y_i = Y_{t_i}$ and $F_i = F_{t_i|t_{i-1}}$. More formally, we let $\hat{Y}_{T+\tau}(T)$ (or $F_{T+\tau|T}$) denote the forecast of a given time series $\{Y_t\}_{t\in\mathbb{Z}^+}$ at time $T+\tau$, where T is a specified origin and $\tau \in \mathbb{Z}^+$. In that setting, the MSE becomes

$$MSE(T) = \frac{1}{T}\sum_{t=1}^{T}(Y_t - \hat{Y}_{(t-1)+1}(t-1))^2$$

Similarly, we define the MAD as

$$MAD(T) = \frac{1}{T}\sum_{t=1}^{T}|Y_t - \hat{Y}_{(t-1)+1}(t-1)|$$

which is the forecast $\hat{Y}_{T+\tau}(T)$ with $T = t-1$ and $\tau = 1$. An approximate 95% prediction interval for $\hat{Y}_{T+\tau}(T)$ is given by

$$\hat{Y}_{T+\tau}(T) \pm z_{0.25}1.25MAD(T)$$

where $z_{0.25} \approx 1.96$.

11.2 Statistical inference

As discussed in Section (11.1), statistical inference is the process of using data analysis to deduce the properties of an underlying probability distribution. Using data drawn from a population with some form of sampling, it makes propositions about a population. It is related to statistical modelling, and, given a hypothesis about a population for which we draw inferences, it consists in (1) selecting a statistical model of the process generating the data, and (2) deducing propositions from that model. Inference can extend to forecasting, prediction and estimation of unobserved

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

values either in or associated with the population being studied. Some common forms of statistical proposition include a point estimate, an interval estimate, a credible interval, the rejection of a hypothesis, clustering or classification of data points into groups.

See textbooks by Cox [2006], Konishi et al. [2008], Freedman [2009], among others.

11.2.1 Modelling time series

Given the time series $(X_t)_{t\in\mathbb{Z}}$ at time t, we can either decompose it into elements and estimate each components separately, or, we can directly model the series with a model such as an autoregressive moving average (ARMA). One can therefore spend his time either modelling each separate element and try to recombine them, or, directly modelling the process X_t.

11.2.1.1 Structural time series

When analysing a time series, the classical approach is to decompose it into components: the trend, the seasonal component, and the irregular term. More generally, the structural time series model proposed by Majani [1987] decomposes the time series $(X_t)_{t\in\mathbb{Z}}$ at time t into four elements:

1. the trend (T_t) : long term movements in the mean
2. the seasonal effects (I_t) : cyclical fluctuations related to the calendar
3. the cycles C_t : other cyclical fluctuations (such as business cycles)
4. the residuals E_t : other random or systematic fluctuations

The idea being to create separate models for these four elements and to combine them either additively

$$X_t = T_t + I_t + C_t + E_t$$

or multiplicatively

$$X_t = T_t \cdot I_t \cdot C_t \cdot E_t$$

which can be obtained by applying the logarithm. Forecasting is done by extrapolating T_t, I_t and C_t, and expecting $E[E_t] = c \in \mathbb{R}$.

11.2.1.2 Some simple statistical models

Rather than modelling the elements of the time series $(X_t)_{t\in\mathbb{Z}}$ we can directly model the series. For illustration purpose we present a few basic statistical models describing the data. Note, each of these models has a number of variants, which are refinements of the basic models.

11.2.1.2.1 AR process An autoregressive (AR) process is one in which the change in the variable at a point in time is linearly correlated with the previous change. In general, the correlation declines exponentially with time and disappear in a relatively short period of time. Letting Y_n be the change in Y at time n, with $0 \leqslant Y \leqslant 1$, then we get

$$Y_n = c_1 Y_{n-1} + ... + c_p Y_{n-p} + e_n$$

where $|c_l| \leqslant 1$ for $l = 1, .., p$, and e is a white noise series with mean 0 and variance σ_e^2. The restrictions on the coefficients c_l ensure that the process is stationary, that is, there is no long-term trend, up or down, in the mean or variance. This is an $AR(p)$ process where the change in Y at time n is dependent on the previous p periods. To test

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

for the possibility of an AR process, a regression is run where the change at time n is the dependent variable, and the changes in the previous q periods (the lags) are used as independent variables. Evaluating the t-statistic for each lag, if any of them are significant at the 5% level, we can form the hypothesis that an AR process is at work.

11.2.1.2.2 MA process In a moving average (MA) process, the time series is the result of the moving average of an unobserved time series

$$Y_n = d_1 e_{n-1} + ... + d_p e_{n-q} + e_n$$

where $|d_l| < 1$ for $l = 1, .., q$. The restriction on the coefficients d_l ensure that the process is invertible. In the case where $d_l > 1$, future events would affect the present, and the process is stationary. Because of the moving average process, there is a linear dependence on the past and a short-term memory effect.

11.2.1.2.3 ARMA process In an autoregressive moving average ($ARMA$) model, we have both some autoregressive terms and some moving average terms which are unobserved random series. We get the general $ARMA(p, q)$ form

$$Y_n = c_0 + c_1 Y_{n-1} + ... + c_p Y_{n-p} - d_1 e_{n-1} - ... - d_q Y_{n-q} + e_n$$

where p is the number of autoregressive terms, q is the number of moving average terms, and e_n is a random variable with a given distribution F and $c_0 \in \mathbb{R}$ is the drift.

11.2.1.2.4 ARIMA process Both AR and $ARMA$ models can be absorbed into a more general class of processes called autoregressive integrated moving average (ARIMA) models which are specifically applied to nonstationary time series. While they have an underlying trend in their mean and variance, by taking successive differences of the data, these processes become stationary. For instance, a price series is not stationary merely because it has a long-term growth component. That is, the price will not tend towards an average value as it can grow without bound. Fortunately, in the efficient market hypothesis (EMH), it is assumed that the changes in price (or returns) are stationary. Typically, price changes are specified as percentage changes, or, log differences, which is the first difference. However, in some series, higher order differences may be needed to make the data stationary. Hence, the difference of the differences is a second-order $ARIMA$ process. In general, we say that Y_t is a homogeneous nonstationary process of order d if

$$Z_t = \Delta^d Y_t$$

is stationary, where Δ represents differencing, and d represents the level of differencing. If Z_t is an $ARMA(p, q)$ process, then Y_t is considered an $ARIMA(p, d, q)$ process. The process does not have to be mixed as if Y_t is an $ARIMA(p, d, 0)$ process, then Z_t is an $AR(p)$ process.

11.2.1.2.5 ARCH process We now introduce popular models to describe the conditional variance of market returns. The basic autoregressive conditional heteroskedasticity ($ARCH$) model developed by Engle [1982] became famous because

- they are a family of nonlinear stochastic processes (as opposed to $ARMA$ models)
- their frequency distribution is a high-peaked, fat-tailed one
- empirical studies showed that financial time series exhibit statistically significant $ARCH$.

In the ARCH model, time series are defined by normal probability distributions but time-dependent variances. That is, the expected variance of a process is conditional on its previous value. The process is also autoregressive in that it has a time dependence. A sample frequency distribution is the average of these expanding and contracting normal distributions, leading to fat-tailed, high-peaked distribution at any point in time. The basic model follows

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} Y_n &= S_n e_n \\ S_n^2 &= \alpha_0 + \alpha_1 e_{n-1}^2 \end{aligned}$$

where e is a standard normal random variable, and α_1 is a constant. Typical values are $\alpha_0 = 1$ and $\alpha_1 = \frac{1}{2}$. Once again, the observed value Y is the result of an unobserved series, e, depending on past realisations of itself. The nonlinearity of the model implies that small changes will likely be followed by other small changes, and large changes by other large changes, but the sign will be unpredictable. Further, large changes will amplify, and small changes will contract, resulting in fat-tailed high-peaked distribution.

11.2.1.2.6 GARCH process Bollerslev [1986] formalised the generalised $ARCH$ (or $GARCH$) by making the S variable dependent on the past as well,

$$\begin{aligned} Y_n &= S_n e_n \\ S_n^2 &= \alpha_0 + \alpha_1 e_{n-1}^2 + \beta_1 S_{n-1}^2 \end{aligned}$$

where the three values range from 0 to 1, but $\alpha_0 = 1$, $\alpha_1 = 0.1$, and $\beta_1 = 0.8$ are typical values. $GARCH$ also creates a fat-tailed high-peaked distribution.

11.2.1.2.7 Example of a financial model The main idea behind (G)ARCH models is that the conditional standard deviations of a data series are a function of their past values. A very common model in financial econometric is the $AR(1) - GARCH(1,1)$ process given by

$$\begin{aligned} r_n &= c_0 + c_1 r_{n-1} + a_n \\ v_n &= \alpha_0 + \alpha_1 e_{n-1}^2 + \beta_1 v_{n-1} \end{aligned}$$

where r_n is the log-returns of the data series for each n, v_n is the conditional variance of the residuals for the mean equation [3]for each n, and c_0, c_1, α_0 and α_1 are known parameters that need to be estimated. The $GARCH$ process is well defined as long as the condition $\alpha_1 + \beta_1 < 1$ is satisfied. If this is not the case, the variance process is non-stationary and we have to fit other processes for conditional variance such as Integrated GARCH (IGARCH) models.

11.2.2 Introducing parametric regression

Given a set of observations, we want to summarise the data by fitting it to a model that depends on adjustable parameters. To do so, we design a merit function measuring the agreement between the data and the model with a particular choice of parameters. We can design the merit function such that either small values represent close agreement (Frequentist), or, by considering probabilities, larger values represent closer agreement (Bayesians). In either case, the parameters of the model are adjusted to find the corresponding extremum in the merit function, providing best-fit parameters. The adjustment process is an optimisation problem which we will treat in Chapter (19). However, in some special cases, specific modelling exist, providing an alternative solution. In any case, the fitting procedure should provide:

1. some parameters
2. error estimates on the parameters, or a way to sample from their probability distribution

[3] $a_n = r_n - c_0 - c_1 r_{n-1}$, $a_n = \sqrt{v_n} e_n$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3. a statistical measure of goodness of fit

Note, in the event where the third item suggests that the model is an unlikely match to the data, then the first two items are probably worthless.

11.2.2.1 Some rules for conducting inference

For a given dataset produced by a randomisation design [4], the randomisation distribution of a statistic (under the null-hypothesis) is defined by evaluating the test statistic for all of the plans that could have been generated by the randomisation design. Different schools (or paradigms) of statistical inference have become established. Some of these paradigms are the classical (or frequentist) paradigm, the Bayesian paradigm, the likelihoodist paradigm, and the AIC-based paradigm. See textbook by Lee [2004]

11.2.2.1.1 Frequentist inference In frequentist inference, randomisation allows inferences to be based on the randomisation distribution rather than a subjective model. The central frequentist idea postulates that, given the details of a null hypothesis, there is an implied population (probability distribution) of possible data sets. If the assumed null hypothesis is correct, the actual, measured, data set is drawn from that population. When the measured data occurs very infrequently in the population, then the hypothesis is rejected. One interpretation of frequentist inference is that it is applicable only in terms of frequency probability, that is, in terms of repeated sampling from a population. Focusing on the distribution of the data sets, they neglect the concept of a probability distribution of the hypothesis (Bayesians). That is, for frequentists, there is no statistical universe of models from which the parameters are drawn (the hypothesis). Instead, they identify the probability of the data given the parameters, as the likelihood of the parameters given data. Parameters derived in this way are called maximum likelihood estimators (MLE).

11.2.2.1.2 Bayesian inference In classical frequentist inference, model parameters and hypotheses are considered to be fixed, and probabilities are not assigned to parameters or hypotheses. An alternative approach is to consider Bayesian inference which refers to statistical inference where uncertainty in inferences is quantified using probability. Practically, it describes degrees of belief using probability theory, and it can assign probabilities to model parameters. Beliefs are positive, integrate to one, and obey probability axioms. A Bayesian approach to learning starts with some a priori knowledge about the model structure and the model parameters. This knowledge is represented by a prior probability distribution over the model structures and parameters. It is then updated by using the data to obtain a posterior probability distribution over the models and their parameters. A **prior probability distribution**, often simply called the prior, of an uncertain quantity is the probability distribution that would express one's beliefs about this quantity before some evidence is taken into account.

11.2.2.1.3 Differences Some major differences between the frequentist and the Bayesian approaches to inference are as follows:

- In a frequentist approach to inference, in general, unknown parameters are often treated as having fixed but unknown values that are not capable of being treated as random variates in any sense. Hence, there is no way that probabilities can be associated with them. In contrast, a Bayesian approach to inference does allow probabilities to be associated with unknown parameters, where these probabilities can sometimes have a frequency probability interpretation as well as a Bayesian one. The Bayesian approach allows these probabilities to have an interpretation as representing one's belief that given values of the parameter are true.
- Even though probabilities are involved in both approaches to inference, the probabilities are associated with different types of things. The result of a Bayesian approach can be a probability distribution for what is known about the parameters given the results of the experiment or study. Alternatively, the result of a frequentist

[4] Randomisation is the process of making something random. In the statistical theory of design of experiments, randomisation involves randomly allocating the experimental units across the treatment groups.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

approach is either a **true or false** conclusion from a significance test, or a conclusion in the form that a given sample-derived confidence interval covers the true value: either of these conclusions has a given probability of being correct, where this probability has either a frequency probability interpretation or a pre-experiment interpretation.

11.2.2.2 The least squares estimator

11.2.2.2.1 Maximum likelihood estimator Given N data points (X_i, Y_i), for $i = 0, .., N-1$, we want to fit a parametric model having M adjustable parameters a_j, $j = 0, .., M-1$, predicting a functional relationship between the measured independent and dependent variables, defined as

$$Y(X) = Y(X|\, a_0, .., a_{M-1})$$

Following the frequentist, given a set of parameters, if the probability of obtaining the data set is too small, then we can conclude that the parameters are unlikely to be right. Assuming that each data point Y_i has a measurement error that is independently random and distributed (i.i.d.), such as a Gaussian distribution, around the true model $Y(X)$, and assuming that the standard deviations σ of these normal distributions are the same for all points, then the probability of obtaining the data set is the product of the probabilities of each point

$$P(\text{ data } | \text{ model }) \propto \prod_{i=0}^{N-1} e^{-\frac{1}{2}(\frac{Y_i - Y(X_i)}{\sigma})^2} \Delta Y$$

Alternatively, calling Bayes' theorem in Equation (11.2.2), we get the posterior distribution

$$P(\text{ model } | \text{ data }) \propto P(\text{ data } | \text{ model })P(\text{ model })$$

where $P(\text{ model }) = P(a_0, .., a_{M-1})$ is the prior probability distribution on all models. The most probable model is to maximise the probability of obtaining the data set above, or equivalently, minimise the negative of its logarithm

$$\sum_{i=0}^{N-1} \frac{1}{2}(\frac{Y_i - Y(X_i)}{\sigma})^2 - N \log \Delta Y$$

which is equivalent to minimising the probability above since N, σ, and ΔY are all constants. In that setting, we recover the least squares fit

$$\text{minimise over } a_0, .., a_{M-1} : \sum_{i=0}^{N-1} (Y_i - Y(X_i|a_0, .., a_{M-1}))^2$$

Under specific assumptions on measurement errors (see above), the least-squares fitting is the most probable parameter set in the Bayesian sense (assuming flat prior), and it is the maximum likelihood estimate of the fitted parameters. Relaxing the assumption of constant standard deviations, by assuming a known standard deviation σ_i for each data point (X_i, Y_i), then the MLE of the model parameters, and the Bayesian most probable parameter set, is given by minimising the quantity

$$\chi^2 = \sum_{i=0}^{N-1} (\frac{Y_i - Y(X_i)}{\sigma_i})^2 \tag{11.2.1}$$

called the chi-square, which is a sum of N squares of normally distributed quantities, each normalised to unit variance. Note, in practice measurement errors are far from Gaussian, and the central limit theorem does not apply, leading to fat tail events skewing the least-squares fit. In some cases, the effect of nonnormal errors is to create an abundance of outlier points decreasing the probability Q that the chi-square should exceed a particular value χ^2 by chance.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.2.2.2.2 Linear models So far we have made no assumption about the linearity or nonlinearity of the model $Y(X|\, a_0, .., a_{M-1})$ in its parameters $a_0, .., a_{M-1}$. The simplest model is a straight line

$$Y(X) = Y(X|\, a, b) = a + bX$$

called linear regression. Assuming that the uncertainty σ_i associated with each measurement Y_i is known, and that the dependent variables X_i are known excatly, we can minimise Equation (11.2.1) to determine a and b. At its minumum, derivatives of $\xi^2(a, b)$ with respect to a and b vanish. See Press et al. [1992] for explicit solution of a and b, covariance of a and b characterising the uncertainty of the parameter estimation, and an estimate of the goodness of fit of the data. We can also consider the general linear combination

$$Y(X) = a_0 + a_1 X + a_2 X^2 + \cdots + a_{M-1} X^{M-1}$$

which is a polynomial of degree $M - 1$. Further, linear combination of sines and cosines is a Fourier series. More generally, we have models of the form

$$Y(X) = \sum_{k=0}^{M-1} a_k \phi_k(X)$$

where the quantities $\phi_0(X), .., \phi_{M-1}(X)$ are arbitrary fixed functions of X called basis functions which can be non-linear (linear refers only to the model's dependence on its parameters a_k). In that setting, the chi-square merit function becomes

$$\chi^2 = \sum_{i=0}^{N-1} \Big(\frac{Y_i - \sum_{k=0}^{M-1} a_k \phi_k(X_i)}{\sigma_i}\Big)^2$$

where σ_i is the measurement error of the ith data point. We can use optimisation to minimise χ^2, or in special cases we can use specific techniques. We let A be an $N \times M$ matrix constructed from the M basis functions evaluated at the N abscissas X_i, and from the N measurement errors σ_i with element

$$A_{ij} = \frac{\phi_j(X_i)}{\sigma_i}$$

This matrix is called the design matrix, and in general $N \geqslant M$. We also define the vector b of length N with element $b_i = \frac{Y_i}{\sigma_i}$, and denote the M vector whose components are the parameters to be fitted $a_0, .., a_{M-1}$ by a. The minimum of the merit function occurs where the derivative of χ^2 with respect to all M parameters a_k vanishes. It yields M equations

$$\sum_{i-0}^{N-1} \frac{1}{\sigma_i}\big(Y_i - \sum_{j=0}^{M-1} a_j \phi_j(X_i)\big)\phi_k(X_i) = 0\,,\, k = 0, .., M-1$$

Interchanging the order of summations, we get the normal equations of the least-squares problem

$$\sum_{j=0}^{M-1} \alpha_{kj} a_j = \beta_k$$

where $\alpha = A^\top.A$ is an $M \times M$ matrix, and $\beta = A^\top.b$ is a vector of length M. In matrix form, the normal equations become

$$(A^\top.A)a = A^\top.b$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

which can be solved for the vector a by LU decomposition, Cholesky decomposition, or Gauss-Jordan elimination (see Press et al. [1992]). The inverse matrix $C = \alpha^{-1}$ is called the covariance matrix, and is closely related to the uncertainties of the estimated parameters a. These uncertainties are estimated as

$$\sigma^2(a_j) = C_{jj}$$

the diagonal elements of C, being the variances of the fitted parameters a. The off-diagonal elements C_{jk} are the covariances between a_j and a_k.

11.2.2.2.3 Nonlinear models In the case where the model depends nonlinearly on the set of M unknown parameters a_k for $k + 0, .., M - 1$, we use the same method as above where we define a χ^2 merit function and determine best-fit parameters by its minimisation. This is similar to the general nonlinear function minimisation problem. If we are sufficiently close to the minimum, we expect the χ^2 function to be well approximated by a quadratic form

$$\chi^2(a) \approx \gamma - d.a + \frac{1}{2}a.D.a$$

where d is an M-vector and D is an $M \times M$ matrix. If the approximation is a good one, we can jump from the current trial parameters a_{cur} to the minimising ones a_{min} in a single leap

$$a_{min} = a_{cur} + D^{-1}.[-\nabla\chi^2(a_{cur})]$$

However, in the where the approximation is a poor one, we can take a step down the gradient, as in the steepest descent method, getting

$$a_{next} = a_{cur} - cst \times \nabla\chi^2(a_{cur})$$

for small constant cst. In both cases we need to compute the gradient of the χ^2 function at any set of parameters a. For a_{min}, we also need the matrix D, which is the second derivative matrix (Hesian matrix) of the χ^2 merit function, at any a. In this particular case, we know exactly the form of χ^2, since it is based on a model function that we specified, so that the Hessian matrix is known to us.

11.2.3 Introducing Bayesian statistics

Bayesian statistics is a theory in the field of statistics based on the Bayesian interpretation of probability, where probability expresses a degree of belief in an event based on prior knowledge about the event, such as the results of previous experiments, or on personal beliefs about the event. Bayes' theorem describes the conditional probability of an event based on data as well as prior information or beliefs about the event or conditions related to the event. This is used to compute and update probabilities after obtaining new data and can directly assign a probability distribution that quantifies the belief to the parameter or set of parameters.

See books by Gelman et al. [1995], Robert [2007], Bernardo et al. [2009], McElreath [2015].

11.2.3.1 Bayesian inference

Given some data x and parameters θ from a model, the **prior probability** is denoted $P(\theta)$. On the other hand, the **posterior probability** of a random event or an uncertain proposition is the conditional probability that is assigned after the relevant evidence or background is taken into account. The posterior probability is the probability of the parameters θ given the evidence (or observation) X, that is, $P(\theta|\, X)$. Given a probability distribution function $P(\theta)$ and the **likelihood function** $P(X|\, \theta)$, the posterior probability is defined as (see Appendix (24.6.1))

$$P(\theta|\, X) = \frac{P(X|\, \theta)}{P(X)}P(\theta)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $P(X)$ is a normalising constant.
Bayesian inference uses the available posterior beliefs as the basis for making statistical propositions. Bayesian inference is calibrated with reference to an explicitly stated utility, or loss function. The Bayes rule is the one which maximises expected utility, averaged over the posterior uncertainty. Formal Bayesian inference therefore automatically provides optimal decisions in a decision theoretic sense.

Bayes's theorem (see Appendix (24.6.1)) relates the conditional probabilities of two events, A and B,

$$P(A|B) = P(A)\frac{P(B|A)}{P(B)} \tag{11.2.2}$$

where $P(A|B)$ is the probability of A given that B has occurred. A and B need not to be repeatable events, and can be propositions or hypotheses, obtaining a set of consistent rules for conducting inference. All Bayesian probabilities are viewed as conditional on some collective background information I. Assuming some hypothesis H (or model M), even before any explicit data exist, we can assign some **degree of plausibility**, $P(H|I)$, called the **Bayesian prior**. When some data set $D_1 \in \mathcal{D}$ comes along, using Equation (11.2.2), we reassess the plausibility of hypothesis H as

$$P(H|D_1I) = P(H|I)\frac{P(D_1|HI)}{P(D_1|I)}$$

where the numerator is calculable as the probability of a data set given the hypothesis (the likelihood function), and the denominator is the prior predictive probability of the data. The latter is a normalisation constant ensuring that the probability of all hypotheses sums to unity. When some additional data D_2 comes along, we can further refine the estimate of the probability of H

$$P(H|D_2D_1I) = P(H|D_1I)\frac{P(D_2|HD_1I)}{P(D_2|D_1I)}$$

and so on. From the product rule for probabilities $P(AB|C) = P(A|C)P(B|AC)$, we get

$$P(H|D_2D_1I) = P(H|I)\frac{P(D_2D_1|HI)}{P(D_2D_1|I)}$$

obtaining the same answer as if all the data D_1D_2 had been taken together.
We can also assume a prior distribution over model structures $P(M)$ and a prior distribution over parameters for each model structure $P(\theta|\ M)$. A data set D is used to form a posterior distribution over models using Bayes rule

$$P(\theta|\ M) = \frac{\int P(D|\ \theta, M)P(\theta|\ M)d\theta P(M)}{P(D)}$$

which averages over the uncertainty in the parameters. For a given model structure, the posterior distribution over the parameters is

$$P(\theta|\ M, D) = \frac{P(D|\ \theta, M)P(\theta|\ M)}{P(D|\ M)}$$

For example, given a time series $Y = (Y_1, ..., Y_T)$, we assume that there are K different candidates models $M_1, .., M_K$ for generating the time series Y. In a Bayesian setting, each of these models is assigned a prior probability $p(M_k)$ with the goal of deriving the posterior model probability $p(M_k|Y)$ (the probability of a hypothesis M_k given the observed evidence Y) for each model M_k, for $k = 1, .., K$.
We can also predict the next observation Y_{T+1} based on the data set and the models. The Bayesian prediction is

$$P(Y_{T+1}|\ Y) = \int P(Y_{T+1}|\ \theta, M, Y)P(\theta|\ M, Y)P(M|\ Y)d\theta dM$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

This is the **predictive distribution** for the model, which averages over both the uncertainty in the model structure and in the parameters. Note, given a model M, we can estimate the parameter vector $\widehat{\theta}$ maximising the likelihood $P(D|\,\theta, M)$. In the limit of a large data set D and a uniform prior over the parameters, the posterior $P(\theta|\,M, D)$ will be sharply peaked around the maxima of the likelihood. Hence, the predictions of a single maximum likelihood model will be similar to those obtained bt Bayesian integration over the parameters.

11.2.3.2 Marginal likelihood function

A marginal likelihood function, or integrated likelihood, is a likelihood function in which some parameter variables have been marginalised. In the context of Bayesian statistics, it may also be referred to as the evidence, or, model evidence. In Bayesian statistics, it is the natural measure of model fit. It is used to evaluate model fit as it quantifies the joint probability of the data under the prior.
Given a set of independent identically distributed (i.i.d.) data points $X = \{x_1, ..., x_n\}$, where $x_i \sim p(x_i|\theta)$, for some probability distribution with parameters θ, which is itself is a random variable described by a distribution $\theta \sim p(\theta|\alpha)$, the marginal likelihood is the conditional probability $p(X|\alpha)$, where θ has been marginalised out (integrated out). In Bayesian statistics, it is given by

$$p(X|\alpha) = \int_\theta p(X|\theta)p(\theta|\alpha)d\theta$$

In classical (frequentist) statistics, the concept of marginal likelihood occurs instead in the context of a joint parameter $\theta = (\psi, \lambda)$, where ψ is the actual parameter of interest, and λ is a non-interesting nuisance parameter. If there exists a probability distribution for λ, it is often desirable to consider the likelihood function only in terms of ψ marginalising out λ

$$\mathcal{L}(\psi; X) = p(X|\psi) = \int_\lambda p(X|\,\lambda, \psi)p(\lambda|\psi)d\lambda$$

However, marginal likelihoods are generally difficult to compute if the likelihood is peaked with respect to the prior (see Robert et al. [2009]). Exact solutions are known for a small class of distributions, particularly when the marginalised-out parameter is the conjugate prior of the distribution of the data. In other cases, some kind of numerical integration method is needed, either a general method such as Gaussian integration, or, a Monte Carlo method, or a method specialised to statistical problems such as the Laplace approximation, Gibbs/Metropolis sampling, or the EM algorithm.

In Bayesian model comparison, the marginalised variables are parameters for a particular type of model, and the remaining variable is the identity of the model itself. In this case, the marginalised likelihood is the probability of the data given the model type, not assuming any particular model parameters. Writing θ for the model parameters, the marginal likelihood for the model M is

$$p(X|M) = \int p(X|\,\theta, M)p(\theta|M)d\theta$$

where $\{p(X|\,\theta, M); \theta \in \Theta\}$ is a likelihood function or sampling distribution. It can be used to calculate the posterior probability of the model given the data $P(M|X) \propto p(X|M)P(X)$, as it is the probability of the data being generated under the prior when the model is correctly specified. We use the term model evidence in this context. It is important because the posterior odds ratio for a model M_1 against another model M_2 involves a ratio of marginal likelihoods, called Bayes factor. It quantifies the prior-to-posterior odds on observing the data:

$$\frac{p(M_1|X)}{p(M_2|X)} = \frac{p(M_1)}{p(M_2)}\frac{p(X|M_1)}{p(X|M_2)}$$

which can be stated schematically as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\text{posterior odds} \;=\; \text{prior odds} \;\times\; \text{Bayes factor}$$

From the property of probability distributions, the log marginal likelihood can be written as the sum of log conditionals

$$\log p(X|M) = \sum_{i=1}^{n} \log p(x_i|\, x_{1:i-1}, M)$$

where

$$p(x_i|\, x_{1:i-1}, M) = \int p(x_i|\, \theta, M) p(\theta|\, x_{1:i-1}, M) d\theta$$

is the posterior predictive probability for $i > 1$, $p(x_1|\, x_{1:0}, M) = \int p(x_1|\, \theta, M) p(\theta|M) d\theta$, and this representation is true for any permutation of the data indices.

11.2.3.3 Variational Bayesian methods

Variational Bayesian methods are a family of techniques for approximating intractable integrals arising in Bayesian inference and machine learning (see Beal [2003]). They are typically used in complex statistical models consisting of observed variables (the data) as well as unknown parameters and latent variables, with various sorts of relationships among the three types of random variables, as might be described by a graphical model. Variational Bayes can be seen as an extension of the EM (expectation-maximisation) algorithm from maximum a posteriori estimation (MAP estimation) of the single most probable value of each parameter to fully Bayesian estimation which computes (an approximation to) the entire posterior distribution of the parameters and latent variables. As in EM, it finds a set of optimal parameter values, and it has the same alternating structure as does EM, based on a set of interlocked (mutually dependent) equations that cannot be solved analytically.

In variational inference, the posterior distribution over a set of unobserved variables $Z = \{z_1, ..., z_n\}$ given some data X, denoted $P(Z|X)$, is approximated by a so-called **variational distribution** $Q(Z)$:

$$P(Z|X) \approx Q(Z) \tag{11.2.3}$$

This approximated distribution is restricted to belong to a family of distributions of simpler form (such as a family of Gaussian distributions) than the posterior $P(Z|X)$, but with the intention of being similar to the true posterior distribution. The similarity (or dissimilarity) is measured in terms of a dissimilarity function $d(Q; P)$. Thus, inference is performed by selecting the distribution $Q(Z)$ that minimises $d(Q; P)$.
The most common type of variational Bayes uses the Kullback-Leibler divergence (KL-divergence) of P from Q as the choice of dissimilarity function, making the minimisation tractable. The KL-divergence (see Definition (2.2.4)) is defined as

$$D_{KL}(Q||P) = \sum_{Z} Q(Z) \log \frac{Q(Z)}{P(Z|X)}$$

Note, Q and P are reversed to be conceptually similar to the expectation-maximisation algorithm.

Variational techniques are typically used to form an approximation for:

$$P(Z|X) = \frac{P(X|Z)P(Z)}{P(X)} = \frac{P(X|Z)P(Z)}{\int_Z P(X,Z)dZ}$$

As discussed in Section (11.2.3.2), the marginalisation over Z to calculate $P(X)$ in the denominator is typically intractable, because, for example, the search space of Z is combinatorially large. Therefore, we seek an approximation,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

using $Q(Z) \approx P(Z|X)$.
Given that $P(Z|X) = \frac{P(X,Z)}{P(X)}$ the KL-divergence above can also be written as

$$D_{KL}(Q||P) = \sum_Z Q(Z)\big[\log \frac{Q(Z)}{P(Z,X)} + \log P(X)\big] = \sum_Z Q(Z)\big[\log Q(Z) - \log P(Z,X)\big] + \sum_Z Q(Z)[\log P(X)]$$

Since the marginal distribution $P(X)$ is a constant with respect to Z, and $\sum_Z Q(Z) = 1$ because $Q(Z)$ is a distribution, we have

$$D_{KL}(Q||P) = \sum_Z Q(Z)\big[\log Q(Z) - \log P(Z,X)\big] + \log P(X)$$

According to the definition of the expected value (for a discrete random variable), the above equation can be written as follows

$$D_{KL}(Q||P) = E_Q[\log Q(Z) - \log P(Z,X)] + \log P(X)$$

We can then get the marginal likelihood

$$\log P(X) = D_{KL}(Q||P) - E_Q[\log Q(Z) - \log P(Z,X)] = D_{KL}(Q||P) + \mathcal{L}(Q) \tag{11.2.4}$$

and

$$\mathcal{L}(Q) = -E_Q[\log \frac{Q(Z)}{P(Z,X)}] = -E_Q[\log \frac{Q(Z)}{P(Z|X)P(X)}]$$

Since the log evidence $\log P(X)$ is fixed with respect to Q, maximising $\mathcal{L}(Q)$ minimises the KL divergence of Q from P. Thus, by choosing appropriately the variational distribution Q, then $\mathcal{L}(Q)$ becomes tractable to compute and to maximise. As a result, we have both an analytical approximation Q for the posterior $P(Z|X)$, and a lower bound $\mathcal{L}(Q)$ for the evidence $\log P(X)$ since the KL-divergence is non-negative.
The lower bound $\mathcal{L}(Q)$ is known as the (negative) variational free energy in analogy with thermodynamic free energy because it can also be expressed as a negative energy $E_Q[\log P(Z,X)]$ plus the entropy of Q. The term $\mathcal{L}(Q)$ is also known as Evidence Lower Bound, abbreviated as ELBO, to emphasise that it is a lower bound on the evidence of the data.

11.2.4 Statistical classification

See books by Mitchell [1997], Jebara [2004], among other.

11.2.4.1 Description

In machine learning and statistics, classification is the problem of identifying to which of a set of categories (subpopulations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. A common subclass of classification is probabilistic classification. Algorithms of this nature use statistical inference to find the best class for a given instance. They output a probability of the instance being a member of each of the possible classes. The best class is normally then selected as the one with the highest probability. These algorithms have numerous advantages over non-probabilistic classifiers:

- They can output a confidence value associated with their choice (in general, a classifier that can do this is known as a confidence-weighted classifier).
- They can abstain when their confidence of choosing any particular output is too low.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Probabilistic classifiers can be more effectively incorporated into larger machine-learning tasks due to the probabilities they generate. As a result, it partially or completely avoids the problem of error propagation.

As discussed in Section (11.2.2.1), there are different schools (or paradigms) of statistical inference such as the frequentist paradigm and the Bayesian paradigm.

- In the former, Fisher [1938] considered the context of two-group problems, which lead to Fisher's linear discriminant function as the rule for assigning a group to a new observation. It assumed that data-values within each of the two groups had a multivariate normal distribution. The extension of this same context to more than two-groups has also been considered with a restriction imposed that the classification rule should be linear. Later work for the multivariate normal distribution allowed the classifier to be non-linear: several classification rules can be derived based on different adjustments of the Mahalanobis distance, with a new observation being assigned to the group whose centre has the lowest adjusted distance from the observation.

- In the latter, the procedures provide a natural way of taking into account any available information about the relative sizes of the different groups within the overall population. They tend to be computationally expensive and, in the days before Markov chain Monte Carlo computations were developed, approximations for Bayesian clustering rules were devised.

11.2.4.2 The generative and discriminative approaches

In statistical classification, including machine learning, the two main approaches are called

1. the generative approach, and
2. the discriminative approach.

These methods compute classifiers by using different approaches, differing in the degree of statistical modelling. In both cases, the notion of probabilities is introduced (see Appendix (24)). Jebara [2004] distinguished three major types:

1. Given an observable variable X and a target variable Y, a generative model is a statistical model of the joint probability distribution on X and Y, denoted $P(X, Y)$.
2. A discriminative model is a model of the conditional probability of the target Y, given an observation x, symbolically, $P(Y|X = x)$.
3. Classifiers computed without using a probability model are also referred to loosely as discriminative.

These three classes are referred to as generative learning, conditional learning, and discriminative learning. However, Ng et al. [2002] only distinguished two classes, calling them generative classifiers (joint distribution) and discriminative classifiers (conditional distribution or no distribution), not distinguishing between the latter two classes. Analogously, a classifier based on a generative model is a generative classifier, while a classifier based on a discriminative model is a discriminative classifier, though this term also refers to classifiers that are not based on a model. Standard examples of each, all of which are linear classifiers, are: generative classifiers: naive Bayes classifier and linear discriminant analysis; discriminative model: logistic regression; non-model classifier: perceptron and support vector machine.

When classifying data, we want to go from an observation x to a label y (or probability distribution on labels). We can compute this directly, without using a probability distribution (distribution-free classifier), or we can estimate the probability of a label given an observation, $P(Y|X = x)$ (discriminative model), and base classification on that, or we can estimate the joint distribution $P(X, Y)$ (generative model), from that compute the conditional probability

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$P(Y|X = x)$, and then base classification on that. These methods are increasingly indirect, but increasingly probabilistic, allowing more domain knowledge and probability theory to be applied. In practice different approaches are used, depending on the particular problem, and hybrids can combine strengths of multiple approaches.
More generally, a generative model is a model of the conditional probability of the observable X, given a target y, symbolically, $P(X|Y = y)$, while a discriminative model is a model of the conditional probability of the target Y, given an observation x, symbolically, $P(Y|X = x)$. Regardless of the precise definition, the terminology is due to the fact that a generative model can be used to "generate" random instances (outcomes), either of an observation and target (x, y) or of an observation x given a target value y, while a discriminative model or discriminative classifier (without a model) can be used to "discriminate" the value of the target variable Y, given an observation x. The difference between "discriminate" (distinguish) and "classify" is subtle, and these are not consistently distinguished.

11.2.4.3 Relationship between models

In application to classification, the observable X is frequently a continuous variable, the target Y is generally a discrete variable consisting of a finite set of labels, and the conditional probability $P(Y|X)$ can also be interpreted as a (non-deterministic) target function $f : X \to Y$, considering X as inputs and Y as outputs.
Given a finite set of labels, the two definitions of "generative model" are closely related. As discussed in Appendix (24.1.2), a model of the conditional distribution $P(X|Y = y)$ is a model of the distribution of each label, and a model of the joint distribution is equivalent to a model of the distribution of label values $P(Y)$, together with the distribution of observations given a label, $P(X|Y)$, symbolically, $P(X, Y) = P(X|Y)P(Y)$. Thus, while a model of the joint probability distribution is more informative than a model of the distribution of label (but without their relative frequencies), it is a relatively small step, hence these are not always distinguished.

Given a model of the joint distribution, $P(X, Y)$, the distribution of the individual variables can be computed as the marginal distributions (or unconditional density of input data) $P(X) = \sum_y P(X, Y = y)$ and $P(Y) = \int_x P(Y, X = x)$ (considering X as continuous, hence integrating over it, and Y as discrete, hence summing over it), and either conditional distribution can be computed from the definition of conditional probability:

$$P(X|Y) = \frac{P(X, Y)}{P(Y)} \text{ and } P(Y|X) = \frac{P(X, Y)}{P(X)} \tag{11.2.5}$$

Given a model of one conditional probability, and estimated probability distributions for the variables X and Y, denoted $P(X)$ and $P(Y)$, one can estimate the opposite conditional probability using Bayes' rule:

$$P(X|Y)P(Y) = P(Y|X)P(X)$$

Thus, given a generative model for $P(X|Y)$ can estimate:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

and given a discriminative model for $P(Y|X)$ can estimate:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

11.2.4.4 Classifying the data

1. A generative algorithm models how the data was generated in order to categorise a signal. Based on our generation assumptions, we want to know which category is most likely to generate this signal. That is, generative algorithms try to learn the joint distribution $P(X, Y)$, which can be transformed into $P(Y|X)$ later to classify the data.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. On the other hand, a discriminative algorithm does not care about how the data was generated, but simply categorises a given signal. So, discriminative algorithms try to learn $P(Y|X)$ directly from the data and then try to classify data.

One of the advantages of generative algorithms is that you can use $P(X,Y)$ to generate new data similar to existing data. On the other hand, discriminative algorithms generally give better performance in classification tasks. However, despite the fact that discriminative models do not need to model the distribution of the observed variables, they cannot generally express complex relationships between the observed and target variables (one need to model the dynamics). In that case, they do not perform better than generative models at classification and regression tasks. The two classes are seen as complementary or as different views of the same procedure (see Bishop et al. [2007]).

11.2.4.5 Types of models

Some types of models for the generative approach and the discriminative approach are listed below.

11.2.4.5.1 Generative models The types of generative models are:

- Gaussian mixture model (and other types of mixture model)
- Hidden Markov model
- Probabilistic context-free grammar
- Bayesian network (Naive bayes, Autoregressive model)
- Averaged one-dependence estimators
- Latent Dirichlet allocation
- Boltzmann machine (Restricted Boltzmann machine, Deep belief network)
- Variational autoencoder (VAE)
- Generative adversarial network (GAN)
- Flow-based generative model

If the observed data are truly sampled from the generative model, then fitting the parameters of the generative model to maximise the data likelihood (see Section (11.2.2.2)) is a common method. However, since most statistical models are only approximations to the true distribution, if we want to infer something about a subset of variables conditional on known values of others, then it can be argued that the approximation makes more assumptions than are necessary to solve the problem at hand. In such cases, it can be more accurate to model the conditional density functions directly using a discriminative model.

11.2.4.5.2 Discriminative models The types of discriminative models are:

- k-nearest neighbours algorithm
- Logistic regression
- Support Vector Machines
- Maximum-entropy Markov models
- Conditional random fields
- Neural networks

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.2.5 Introducing state-space models

11.2.5.1 The state-space form

The state-space form of time series models represent the actual dynamics of a data generation process (see Appendix (23.4.3)). We let Y_t denote the observation from a time series at time t, related to a vector α_t, called the state vector, which is possibly unobserved (hidden) and whose dimension m is independent of the dimension n of Y_t.
The general form of a linear state-space model, is given by the following two equations

$$\begin{aligned} Y_t &= Z_t\alpha_t + d_t + G_t\epsilon_t \,, t = 1, .., T \\ \alpha_{t+1} &= T_t\alpha_t + c_t + H_t\epsilon_t \end{aligned} \tag{11.2.6}$$

where Z_t is an $(n \times m)$ matrix, d_t is an $(n \times 1)$ vector, G_t is an $(n \times (n+m))$ matrix, T_t is an $(m \times m)$ matrix, and H_t is an $(m \times (n+m))$ matrix. The process ϵ_t is an $((n+m) \times 1)$ vector of serially independent, identically distributed disturbances with $E[\epsilon_t] = 0$ and $Var(\epsilon_t) = I$ the identity matrix. We let the initial state vector α_1 be independent of ϵ_t at all time t. The first equation is the observation, or measurement equation, and the second equation is the transition equation. This system of equations is to be compared with the system in Equation (23.4.12) where $u(t)$ is the input vector, $x(t)$ is the state vector (here α_t), and $y(t)$ is the output vector.
The general (first-order Markov) state equation takes the form

$$\alpha_t = f(\alpha_{t-1}, \theta_{t-1}) + \eta_{t-1}$$

where θ is a vector of parameters, and f is a particular function. The general observation equation takes the form

$$Y_t = h(\alpha_t, \theta_t) + \epsilon_t$$

with independent error processes $\{\eta_t\}$ and $\{\epsilon_t\}$. Again, this system of equations is to be compared with the non-linear system in Equation (23.4.13). If the system of matrices do not evolve with time, the state-space model is called time-invariant or time-homogeneous. If the disturbances ϵ_t and initial state vector α_1 are assumed to have a normal distribution, then the model is termed Gaussian. Further, if $G_tH_t^\top = 0$, for all t, then the measurement and transition equations are uncorrelated.

11.2.5.1.1 The inference mechanism The fundamental inference mechanism is Bayesian (see Section (11.2.2.1)) and consists in computing the posterior quantities of interest sequentially in the following recursive calculation:

1. Letting $\psi_t = \{Y_1, .., Y_t\}$ be the information set up to time t, we get the **prior distribution**

$$p(\alpha_t|\psi_{t-1}) = \int p(\alpha_t|\alpha_{t-1})p(\alpha_{t-1}|\psi_{t-1})d\alpha_{t-1}$$

   corresponding to the distribution of the parameters before any data is observed. Note, α is the model hypothesis.

2. Then, the **updating equation** becomes

$$p(\alpha_t|\psi_t) = \frac{p(Y_t|\alpha_t)p(\alpha_t|\psi_{t-1})}{p(Y_t|\psi_{t-1})}$$

   where the **sampling distribution**, $p(Y_t|\alpha_t)$, is the distribution of the observed data conditional on its parameters. Note, $p(Y_t|\alpha_t)$ is called the **marginal likelihood** for model α (it is the probability of observing Y given α).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The updates provides an analytical solution if all densities in the state and observation equation are Gaussian, and both the state and the observation equation are linear. If these conditions are met, the Kalman filter (see Section (11.2.5.2)) provides the optimal Bayesian solution to the tracking problem. Otherwise we require approximations such as the Extended Kalman filter (EKF), or Particle filter (PF), which approximates non-Gaussian densities and non-linear equations. The Particle filter uses Monte Carlo methods, in particular Importance sampling, to construct the approximations (see Section (12.6.1.3)).

11.2.5.1.2 Some problems Some of the main inference problems include prediction, filtering and smoothing (see Kalman [1960], Rabiner [1989]). If we let $\psi_t = \{Y_1, .., Y_t\}$ denote the information set up to time t, then the problems of interest are:

1. The probability of an observed sequence: Given a particular model α, the probability of a particular output sequence ψ_t is given by

$$P(\psi_t) = \sum_{k=1}^{t} P(\psi_t|\ \alpha_k)P(\alpha_k)$$

   where the sum runs over all possible hidden state sequences. Applying the principle of dynamic programming, one can solve this problem using the forward algorithm.

2. The problem of prediction: It is concerned with computing $E[\alpha_{t+1}|\psi_t]$. Given the model's parameters and a sequence of observations ψ_t, prediction compute the distribution over hidden states of the next latent variable of a sequence, that is, $P(\alpha_{t+1}|\ \psi_t)$.

3. The problem of filtering: It is concerned with calculating $E[\alpha_t|\ \psi_t]$. Given the model's parameters and a sequence of observations ψ_t, filtering compute the distribution over hidden states of the last latent variable at the end of the sequence, that is, $P(\alpha_t|\ \psi_t)$. This problem can be efficiently solved with the forward algorithm.

4. The problem of smoothing: It is concerned with estimating $E[\alpha_t|\psi_T]$ for all $t < T$. It asks about the distribution of the latent variable α_t for $t < T$. It can be seen as the probability distribution over hidden states for a point at time t in the past relative to time T. The forward-backward algorithm is a good method for solving the smoothed values for all hidden state variables.

5. Finding the most likely explanation: the task asks about the joint probability of the entire sequence of hidden states $\alpha_1, ..., \alpha_t$ that generated a particular sequence of observations ψ_t. An example is part-of-speech tagging, where the hidden states represent the underlying parts of speech corresponding to an observed sequence of words. Here, we are interested in the entire sequence of parts of speech, rather than just the part of speech for a single word (as filtering or smoothing would compute). This task requires finding a maximum over all possible state sequences, and can be solved with the Viterbi algorithm.

11.2.5.2 The Kalman filter

The Kalman filter is used for prediction, filtering and smoothing (see Kalman [1960]). In the Linear Gaussian State-Space model, we assume $G_t H_t^\top = 0$ and drop the terms d_t and c_t from the observation and transition equations (11.2.6). Further, we let

$$G_t G_t^\top = \Sigma_t\ ,\ H_t H_t^\top = \Omega_t$$

and the Kalman filter recursively computes the quantities

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$
\begin{aligned}
a_{t|t} &= E[\alpha_t|\psi_t] \text{ filtering} \\
a_{t+1|t} &= E[\alpha_{t+1}|\psi_t] \text{ prediction} \\
P_{t|t} &= MSE(\alpha_t|\psi_{t-1}) \\
P_{t+1|t} &= MSE(\alpha_{t+1}|\psi_t)
\end{aligned}
$$

where MSE is the mean-square error or one-step ahead prediction variance. Then, starting with $a_{1|0}$, $P_{1|0}$, then $a_{t|t}$ and $a_{t+1|t}$ are obtained by running for $t = 1, .., t$ the recursions

$$
\begin{aligned}
V_t &= Y_t - Z_t a_{t|t-1} \text{ , } F_t = Z_t P_{t|t-1} Z_t^\top + \Sigma_t \\
a_{t|t} &= a_{t|t-1} + P_{t|t-1} Z_t^\top F_t^{-1} V_t \\
P_{t|t} &= P_{t|t-1} - P_{t|t-1} Z_t^\top F_t^{-1} Z_t P_{t|t-1} \\
a_{t+1|t} &= T_t a_{t|t} \\
P_{t+1|t} &= T_t P_{t|t} T_t^\top + \Omega_t
\end{aligned}
$$

where V_t denotes the one-step-ahead error in forecasting Y_t conditional on the information set at time $(t-1)$ and F_t is its MSE. The quantities $a_{t|t}$ and $a_{t|t-1}$ are optimal estimators of α_t conditional on the available information. The resulting recursions for $t = 1, .., T-1$ follows:

$$
\begin{aligned}
a_{t+1|t} &= T_t a_{t|t-1} + K_t V_t \\
K_t &= T_t P_{t|t-1} Z_t^\top F_t^{-1} \\
P_{t+1|t} &= T_t P_{t+1|t} L_t^\top + \Omega_t \\
L_t &= T_t - K_t Z_t
\end{aligned}
$$

11.2.5.2.1 Parameter estimation Another application of the Kalman filter is the estimation of any unknown parameters θ that appear in the system matrices. The likelihood for data $Y = (Y_1, .., Y_T)$ can be constructed as

$$
p(Y_1, .., Y_T) = p(Y_T|\ \psi_{T-1}) \cdots p(Y_2|\ \psi_1) p(Y_1) = \prod_{t=1}^{T} p(Y_t|\ \psi_{t-1})
$$

where $\psi_t = \{Y_1, .., Y_t\}$. Assuming that the state-space model is Gaussian, by taking conditional expectations on both sides of the observation equation, with $d_t = 0$, we deduce that, for $t = 1, .., T$,

$$
\begin{aligned}
E[Y_t|\ \psi_{t-1}] &= Z_t a_{t|t-1} \\
Var(Y_t|\ \psi_{t-1}) &= F_t
\end{aligned}
$$

the one-step-ahead prediction density $p(Y_t|\psi_{t-1})$ is the density of a multivariate normal random variable with mean $Z_t a_{t|t-1}$ and and covariance matrix F_t. Thus, the log-likelihood function is given by

$$
\log L = -\frac{nT}{2} \log 2\pi - \frac{1}{2} \sum_{t=1}^{T} \log det F_t - \frac{1}{2} \sum_{t=1}^{T} V_t^\top F_t^{-1} V_t
$$

where $V_t = Y_t - Z_t a_{t|t-1}$. Numerical procedures are used in order to maximise the log-likelihood to obtain the ML estimates of the parameters θ which are consistent and asymptotically normal. If the state-space model is not Gaussian, the likelihood can still be constructed in the same way using the minimum mean square linear estimators of the state vector. However, the estimators $\hat{\theta}$ maximising the likelihood are the quasi-maximum likelihood (QML) estimators of the parameters. They are also consistent and asymptotically normal.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.2.5.2.2 Smoothing Smoothing is another application of Kalman filter where, given a fixed set of data, estimates of the state vector are computed at each time t in the sample period taking into account the full information set available. The algorithm computes $a_{t|T} = E[\alpha_t|\psi_T]$ along with its MSE, and $P_{t|T}$ computed via a set of backward recursions for all $t = 1, .., T-1$. To obtain $a_{t|T}$ and $P_{t|T}$ we start with $a_{T|T}$ and $P_{T|T}$ and run backwards for $t = T-1, .., 0$

$$a_{t|T} = a_{t|t} + P_t^*(a_{t+1|T} - a_{t+1|t})$$
$$P_{t|T} = P_{t|t} + P_t^*(P_{t+1|T} - P_{t+1|t})P_t^* \ , \ P_t^* = P_t T_t^\top P_{t+1|t}$$

the extensive use of the Markov chain Monte Carlo (MCMC), in particular the Gibbs sampler, has given rise to another smoothing algorithm called the simulation smoother and is also closely related to the Kalman filter. In contrast, to the fixed interval smoother, which computes the conditional mean and variance of the state vector at each time t in the sample, a simulation smoother is used for drawing samples from the density $p(\alpha_0, ..., \alpha_T|\ Y_T)$. The first simulation smoother is based on the identity

$$p(\alpha_0, ..., \alpha_T|\ Y_T) = p(\alpha_T|\ Y_T) \prod_{t=0}^{T-1} p(\alpha_t|\ \psi_t, \alpha_{t+1})$$

and a draw from $p(\alpha_0, ..., \alpha_T|\ Y_T)$ s recursively constructed in terms of α_t. Starting with a draw $\hat{\alpha}_T \sim N(\alpha_{T|T}, P_{T|T})$, the main idea is that for a Gaussian state space model $p(\alpha_t|\psi_t, \alpha_{t+1})$ is a multivariate normal density and hence it is completely characterized by its first and second moments. The usual Kalman filter recursions are run, so that $\alpha_{t|t}$ is initially obtained. Then, the draw $\hat{\alpha}_{t+1} \sim p(\alpha_{t+1}|\ \psi_t, \alpha_{t+2})$ is treated as m additional observations and a second set of m Kalman filter recursions is run for each element of the state vector $\hat{\alpha}_{t+1}$. However, the latter procedure involves the inversion of system matrices, which are not necessarily non-singular.

11.2.5.3 Model specification

While state space models are widely used in time series analysis to deal with processes gradually changing over time, model specification is a challenge for these models as one has to specify which components to include and to decide whether they are fixed or time-varying. It leads to testing problems which are non-regular from the view-point of classical statistics. Thus, a classical approach toward model selection which is based on hypothesis testing such as a likelihood ratio test or information criteria such as AIC or BIC cannot be easily applied, because it relies on asymptotic arguments based on regularity conditions that are violated in this context.
For example, we consider the time series $Y = (Y_1, ..., Y_T)$, for $t = 1, ..., T$, modelled with the linear trend model

$$Y_t = \mu_t + \epsilon_t \ , \epsilon_t \sim N(0, \sigma_\epsilon^2)$$

where μ_t is a random walk with a random drift starting from unknown initial values μ_0 and a_0

$$\mu_t = \mu_{t-1} + a_{t-1} + \omega_{1t} \ , \omega_{1t} \sim N(0, \theta_1)$$
$$a_t = a_{t-1} + \omega_{2t} \ , \omega_{2t} \sim N(0, \theta_2)$$

In order to decide whether the drift a_t is time-varying or fixed we could test $\theta_2 = 0$ versus $\theta_2 > 0$. However, it is a non-regular testing problem since the null hypothesis lies on the boundary of the parameter space. Testing the null hypothesis $a_0 = a_1 = ... = a_T$ versus the alternative a_t follows a random walk is, again, non-regular because the size of the hypothesis increases with the number of observations.
One possibility is to consider the Bayesian approach when dealing with such non-regular testing problems. We assume that there are K different candidates models $M_1, .., M_K$ for generating the time series Y. In a Bayesian setting, each of these models is assigned a prior probability $p(M_k)$ with the goal of deriving the posterior model probability $p(M_k|Y)$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(the probability of a hypothesis M_k given the observed evidence Y) for each model M_k, for $k = 1, .., K$.
One strategy for computing the posterior model probabilities is to determine the posterior model probabilities of each model separately by using Bayes' rule $p(M_k|Y) \propto p(Y|M_k)p(M_k)$ where $p(Y|M_k)$ is the marginal likelihood for model M_k (it is the probability of observing Y given M_k). An explicit expression for the marginal likelihood exists only for conjugate problems like linear regression models with normally distributed errors, whereas for more complex models numerical techniques are required. For Gaussian state space models, marginal likelihoods have been estimated using methods such as importance sampling, Chib's estimator, numerical integration and bridge sampling.
The modern approach to Bayesian model selection is to apply model space MCMC methods by sampling jointly model indicators and parameters, using for instance the reversible jump MCMC algorithm (see Green [1995]) or the stochastic variable selection approach (see George and McCulloch [1993] [1997]). The stochastic variable selection approach applied to model selection for regression models aims at identifying non-zero regression effects and allows parsimonious covariance modelling for longitudinal data. Fruhwirth-Schnatter et al. [2010a] considered the variable selection approach for many model selection problems occurring in state space modelling. In the above example, they used binary stochastic indicators in such a way that the unconstrained model corresponds to setting all indicators equal to 1. Reduced model specifications result by setting certain indicators equal to 0.

11.3 SVM models

In machine learning, support vector machines (SVMs) are supervised learning models with associated learning algorithms that analyse data used for classification (see Vapnik [1995-2000]) and regression analysis (see Drucker et al. [1996]).

Vapnik et al. [1963] introduced the Generalized Portrait algorithm which was further developed by Vapnik et al. [1964]. The algorithm implemented by support vector machines is a nonlinear generalisation of the Generalized Portrait algorithm. SVM was extended by Boser et al. [1992] who suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes. Cortes et al. [1995] introduced the soft margin classifier. In the same year the algorithm was extended to the case of regression by Vapnik [1995-2000].

SVM is a learning system using a high dimensional feature space where points are classified by means of assigning them to one of the two disjoint half spaces, either in the pattern space or in a higher dimensional feature space. The main objective is to identify maximum margin hyper plane, that is, the margin of separation between positive and negative examples is maximised. SVM finds the maximum margin hyper plane as the final decision boundary.

11.3.1 An optimisation problem

Assume that $x_i \in \mathbb{R}^d$, $i = 1, ..., N$ forms a set of input vectors with corresponding class labels (binary variables) $y \in \{+1, -1\}$, $i = 1, ..., N$. We want to find the maximum margin hyperplane that divides the group of points x_i for which $y_i = 1$ from the group of points for which $y_i = -1$, which is defined so that the distance between the hyperplane and the nearest point x_i from either group is maximised.
Any hyperplane can be written as the set of points x satisfying

$$w \cdot x - b = 0$$

where w is the normal vector to the hyperplane. The parameter

$$\frac{b}{\|w\|} \tag{11.3.7}$$

determines the offset of the hyperplane from the origin along the normal vector w. If the training data is linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the margin, and the maximum

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

margin hyperplane is the hyperplane lying halfway between them. A linear classifier can be considered as a hyperplane with normal vector $w \in \mathbb{R}^d$ and offset b. The classification of x_i is determined by the hyperplane:

$$\hat{y}_i = \text{sgn}\big(w \cdot x_i + b\big)$$

where

$$\text{sgn}(x) = \begin{cases} -1 \text{ if } x < 0 \\ 0 \text{ if } x - 0 \\ 1 \text{ if } x > 0 \end{cases}$$

To prevent the data points from falling into the margin, we need to impose the following constraints: For each i either

$$w \cdot x_i - b \geqslant 1 \text{ if } y_i = 1$$

or

$$w \cdot x_i - b \leqslant -1 \text{ if } y_i = -1$$

That is, each data point must lie on the correct side of the margin. It can be rewritten as

$$y_i(w \cdot x_i - b) \geqslant 1 \text{ , } \forall 1 \leqslant i \leqslant N$$

An illustration is given in Figure (11.2).

Figure 11.2: Illustration of SVM

The objective of a linear classifier is to find the hyperplane that optimally separates the positive samples from negative ones. This is done by maximising the margin (gain) between the two regions. This as an optimisation problem where we maximise the margin M:

$$\max_{v,M} M$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

over the normal vector v parametrising the decision boundary such that $\|v\| = 1$ and $y \cdot v^\top x_i \geqslant M$. In that setting, the max-margin hyperplane is completely determined by those x_i which lie nearest to it. Hence, these x_i are called support vector.
We let $w = v \cdot M$ and make the substitution

$$\max_{w,M} M$$

such that $\|w\| = \frac{1}{M}$ and $\forall i : y_i \cdot w^\top x_i \geqslant 1$.
Given the inverse relationship between the norm $\|w\|$ and the margin M, maximising the distance between the planes is equivalent to minimising $\|w\|$. We can then transform this ratio into a minimisation problem

$$\min_{w} \|w\|^2$$

such that $\forall i : y_i \cdot w^\top x_i \geqslant 1$. It is a convex optimisation problem with constraints. However, this method only works if the data is linearly separable. Thus, to deal with non-separable data we need to introduce slack variables into each constraint: $\forall i : y_i \cdot w^\top x_i \geqslant 1 - \epsilon_i$, getting soft-margin SVM. To avoid the domination of the slack variables we introduce a penalty factor to the objective function, getting

$$\min_{w,\epsilon} \|w\|^2 + C \sum_i \epsilon_i$$

such that $\forall i : y_i \cdot w^\top x_i \geqslant 1 - \epsilon_i$ and $\epsilon_i \geqslant 0$ and $\lambda = \frac{1}{C}$. This this the primal offline SVM. It can be formulated with a new objective function to minimise the hinge loss as follows:

$$\min_{w} \|w\|^2 + C \sum_i \max\big(0, 1 - y_i w^\top x_i\big) \tag{11.3.8}$$

where y_i is the ith target and $w \cdot x$ is the current output. It is a trade off where the first term keep the weights small, while the second term is a sum of hinge loss functions, which are high for a poor fit. Thus, computing the (soft-margin) SVM classifier amounts to maximising the margins, or, minimising the empirical risk function (see Section (10.1.2))

$$f(w,b) = \Big[\frac{1}{N} \sum_{i=1}^{N} \max\big(0, 1 - y_i(w \cdot x_i - b)\big)\Big] + \lambda \|w\|^2 \tag{11.3.9}$$

with hinge loss function

$$L(w; x_i, y_i) = \max\big(0, 1 - y_i(w \cdot x_i - b)\big)$$

The parameter λ determines the tradeoff between increasing the margin size and ensuring the the x_i lie on the correct side of the margin. So, for sufficiently small values of λ, the second term of the equation becomes negligible, and we recover the hard-margin SVM, if the data is linearly separable.
As discussed in Section (10.1.2), the SVP problem is equivalent to empirical risk minimisation (ERM) with Tikhonov regularisation, where the loss function is the hinge loss $L(x, y) = \max(0, xy)$. Thus, SVM is closely related to other fundamental classification algorithms, where the main difference is in the choice of the loss function.

11.3.2 The algorithms

In learning, we want to minimise that objective function. Most of the algorithms are iterative in nature and will hopefully converge to the optimal value. In order to find the break-even point between the algorithms we want to focus on the generalisation error, that is, the expected error on test set.
We can use online convex programming to optimise that problem, getting the Online SVM. Alternatively, we can consider offline algorithm and apply the Lagrange multiplier or the kernel trick.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In the former, from the representation property, at optimal solution, the weight vector w is given by $w = \sum_i \alpha_i y_i x_i$. In the latter, SVM can map the input vectors $x_i \in \mathbb{R}^d$ into a high dimensional feature space $\Phi(x_i) \in H$. A kernel function $k(x_i, x_j)$ performs the mapping $\phi(\cdot)$, that is, $k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$. The value w, in the transformed space, is given by $w = \sum_{i=1}^N \alpha_i y_i \phi(x_i)$. Dot products with w for classification can again be computed by the kernel trick as

$$w \cdot \phi(x) = \sum_{i=1}^N \alpha_i y_i k(x, x_i)$$

We are now present more formally some online and offline algorithms to solve the optimisation problem in Equation (11.3.8).

11.3.2.1 Online algorithms

11.3.2.1.1 A simple algorithm We can devise a simple online algorithm based on the gradient gradient descent (see Section (10.1.2.5)). Given the feasible set of the hyperplane $\mathcal{W} = \{w : \|w\|_2 \leqslant \lambda\}$ and the hinge loss function we get the online algorithm. See Algorithm (26) with learning rate $\eta_i = \frac{1}{\sqrt{i}}$ or $\eta_i = \frac{1}{i}$.

Algorithm 26 SVM Online Algorithm

Require: : Import input vectors and corresponding labels.
1: Pick $w_1 \in \mathcal{W}$ arbitrarily.
2: **for** $i = 1$ to N **do**
3: The incurred loss is $L(w_i) \equiv hinge(w_i; x_i, y_i)$
4: Step forward on the gradient direction: $\widehat{w}_{i+1} = w_i - \eta_i \nabla L(w_i)$
5: Project $\widehat{w}_{i+1}$ back to the feasible set: $w_{i+1} = Proj_{\mathcal{W}}(\widehat{w}_{i+1})$
6: **end for**
7: **Return w**

Since the loss function is not differentiable we can use a sub-gradient to compute the gradient (see discussion below). We now briefly describe the projection (see Section (18.3.2)). Given the feasible set $\mathcal{W}$, the projection from $\widehat{w} \notin \mathcal{W}$ to its nearest point in $\mathcal{W}$ can be performed by multiplying $\widehat{w}$ with a scalar

$$w_{i+1} = Proj(\widehat{w}_{i+1}) = \frac{\lambda \cdot \widehat{w}_{i+1}}{\|\widehat{w}_{i+1}\|}$$

This a direct consequence from the parameter determining the offset hyperplane in Equation (11.3.7). In the case where $\widehat{w} \in \mathcal{W}$, then $Proj(\widehat{w}) = \widehat{w}$.

11.3.2.1.2 More complex algorithms Recent algorithms for finding the SVM classifier include sub-gradient descent (see Shalev-Shwartz et al. [2011]) and coordinate descent (see Hsieh et al. [2008]). Both techniques have proven to offer significant advantages over the traditional approach when dealing with large, sparse datasets. Sub-gradient methods are especially efficient when there are many training examples, and coordinate descent when the dimension of the feature space is high.
Sub-gradient descent algorithms for the SVM work directly with the expression $f(w, b)$ in Equation (11.3.9). Since f is a convex function, traditional gradient descent (or SGD) methods can be adapted, where instead of taking a step in the direction of the functions gradient, a step is taken in the direction of a vector selected from the function's sub-gradient (see Section (18.3.2))).
The coordinate descent algorithms apply to the dual problem. The coefficient α_i, $i = 1, ..., N$, is adjusted in the direction of $\frac{\partial L}{\partial \alpha_i}$. The resulting vector of coefficients $(\alpha_1^{'}, ..., \alpha_N^{'})$ is projected onto the nearest vector of coefficients that satisfies some Euclidean distances. The process is repeated until a near-optimal vector is obtained.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.3.2.1.3 Parallel algorithm Zinkevich et al. [2009] proposed a parallel algorithm for Online SVM by introducing delayed updates. At the ith step the fetched gradient $\nabla L(w_{i-\tau})$ is the result at the τ-th previous step. They proved convergence. See Algorithm (27).

Algorithm 27 SVM Parallel Online Algorithm

Require: : Import input vectors and corresponding labels.
1: Pick $w_1 \in \mathcal{W}$ arbitrarily.
2: **for** $i = 1$ to N **do**
3: The incurred loss is $L(w_i) \equiv hinge(w_i; x_i, y_i)$
4: Step forward on the gradient direction: $\widehat{w}_{i+1} = w_i - \eta_i \nabla L(w_{i-\tau})$
5: Project $\widehat{w}_{i+1}$ back to the feasible set: $w_{i+1} = Proj_{\mathcal{W}}(\widehat{w}_{i+1})$
6: **end for**
7: **Return w**

11.3.2.2 Offline algorithms

11.3.2.2.1 The Lagrange multiplier As discussed in Section (18.5), the primal is a constrained optimisation problem with a differentiable objective function. By solving for the Lagrangian dual of the above problem, one obtains the simplified (unconstrained) optimisation problem called the dual problem. Since minimising $\|w\|^2$ is equivalent to minimising $\frac{1}{2}\|w\|^2$ we can introduce the Lagrange multipliers α_i. We can then write the objective function given in Equation (11.3.8) as

$$L(w, \alpha) = \frac{1}{2}\|w\|^2 - \sum_i^N \alpha_i \big(y_i w^\top x_i - 1\big)$$

where the new objective is

$$\min_w \max_\alpha L(w, \alpha)$$

We can then compute the KKT (Karush-Kuhn-Tucker) conditions for differentiable convex programs (see Section (18.5.1)). Applying the KKT theorem to the SVM optimisation problem we get the dual problem:

1. $\frac{\partial}{\partial w} L(w, \alpha) = 0 \rightarrow w = \sum_i \alpha_i y_i x_i$
2. $y_i w^\top x_i - 1 \geqslant 0$
3. $\sum_i \big(y_i w^\top x_i - 1\big) = 0$

From these conditions, we can see that

1. w can be represented as a linear combination of data points.
2. All data points are at least a normalised distance of 1 from the separating hyperplane.
3. As $\alpha_i \geqslant 0$ either $\alpha_i = 0$ or $y_i w^\top x_i = 1$ for all i.

Thus, the points for which $\alpha_i > 0$ are supporting the hyperplane, and as such are called support vectors. Hence, we write the set of support vectors as

$$\mathcal{S} = \{x_i : y_i w^\top x_i = 1\}$$

We can then substitute $w = \sum_i \alpha_i y_i x_i$ into the Lagrangian and get a simplified objective function

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} L(\alpha) &= \frac{1}{2}\big(\sum_i \alpha_i y_i x_i\big)\big(\sum_j \alpha_j y_j x_j\big) - \sum_i \alpha_i \Big[y_i \big(\sum_j \alpha_j y_j x_j\big)^\top x_i - 1 \Big] \\ &= \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^\top x_j - \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^\top x_j + \sum_i \alpha_i \\ &= \sum_i^N \alpha_i - \frac{1}{2}\sum_{i,j}^N \alpha_i \alpha_j y_i y_j x_i^\top x_j \end{aligned} \tag{11.3.10}$$

subject to $\sum_{i=1}^N \alpha_i y_i = 0$.
If we have α, we can then solve for w, and only need to solve for

$$\alpha^* = \arg\max_\alpha L(\alpha)$$

such that $\alpha_i \geqslant 0$, $\forall i$.

***Remark* 11.3.1** *The objective function only depends on inner products $x_i^\top x_j$, which is very useful for nonlinear classification tasks.*

This inner product can be replaced by a kernel function $k(x_i, x_j)$ which takes the inner product to a higher dimensional space. This is the basis of the kernel trick discussed below.

11.3.2.2.2 The kernel trick We can also apply the kernel trick. In some cases, for the data to be linearly separable we first need to apply a nonlinear transformation to a higher dimensional space. For instance, suppose that we would like to learn a nonlinear classification rule which corresponds to a linear classification rule for the transformed data points $\phi(x_i)$. However, when the dimension of $\phi(\cdot)$ becomes very large we should not use an explicit transformation. Note, if $\phi(x)$ is all ordered monomials of degree d, then $\phi^\top(x)\phi(x^{'}) = (x^\top x^{'})^d$ and we can implicitly work in a higher dimensional space by using a different dot product. This is called the kernel trick and

$$\phi^\top(x)\phi(x^{'}) = k(x, x^{'}) \tag{11.3.11}$$

is a kernel function (see Appendix (23.2.1.4)).
As a result of the representation property (see the dual problem above), at optimal solution, the weight vector w is a linear combination of the data points given by

$$w = \sum_i \alpha_i y_i x_i \tag{11.3.12}$$

Suppose we are given a kernel function defined as above. Then, the classification vector w, in the transformed space, satisfies

$$w = \sum_{i=1}^N \alpha_i y_i \phi(x_i)$$

Note that w can be an infinite dimensional vector (a function). We can treat the problem as a parameter estimation problem.
From Equation (11.3.11), the resulting decision boundary is

$$f(x) = \mathrm{sgn}\big(w \cdot \phi(x) + b\big) = \mathrm{sgn}\big(\sum_{i=1}^N y_i \alpha_i \cdot k(x, x_i) + b\big)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Given the dual problem in Equation (11.3.10), in order to estimate the values α_i, $i = 1, ..., N$ we solve the following quadratic programming problem (see Appendix (18.6))

$$\begin{aligned}
&\max_{\alpha_i}\Big(\sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N\sum_{i=j}^N \alpha_i\alpha_j \cdot y_i y_j \cdot k(x_i, x_j)\Big)\\
&\text{subject to } 0 \leqslant \alpha_i \leqslant c\\
&\sum_{i=1}^N y_i\alpha_i = 0\,,\, i = 1, ..., N
\end{aligned}$$

where the parameter c controls the tradeoff between the margin and misclassification error. In general it is set to $\frac{1}{2N\lambda}$. We can find some index i such that $0 < \alpha_i < c$ so that $\phi(x_i)$ lies on the boundary of the margin in the transformed space, and then solve

$$\begin{aligned}
b = w \cdot \phi(x_i) - y_i &= \big(\sum_{k=1}^N \alpha_k y_k \phi(x_k) \cdot \phi(x_i)\big) - y_i\\
&= \big(\sum_{k=1}^N \alpha_k y_k k(x_k, x_i)\big) - y_i
\end{aligned}$$

Then, new points can be classified by computing

$$z \to \text{sgn}(w \cdot \phi(z) - b) = \text{sgn}\Big(\big(\sum_{i=1}^N \alpha_i y_i k(x_i, z)\big) - b\Big)$$

We can also apply the same method to the primal optimisation problem. Replacing the weight vector w given in Equation (11.3.12) in the primal in Equation (11.3.8), the optimisation problem becomes equivalent to

$$\min_\alpha \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^\top x_j + C\sum_i \max\big(0, 1 - y_i\sum_j \alpha_j y_j x_j^\top x_i\big)$$

Assuming that $w \in \text{Span}\{x_i, \forall i\}$, we replace $x_i^\top x_j$ with $k(x_i, x_j)$. We then let $\tilde{\alpha}_i = \alpha_i y_i$ (since we are not constrained we can flip signs arbitrarily) and obtain the objective function

$$\min_{\tilde{\alpha}} \frac{1}{2}\sum_{i,j}\tilde{\alpha}_i\tilde{\alpha}_j k(x_i, x_j) + C\sum_i \max\big(0, 1 - y_i\sum_j \tilde{\alpha}_j k(x_i, x_j)\big)$$

Using matrix notation, we rewrite the optimisation problem as

$$\min_{\tilde{\alpha}} \frac{1}{2}\tilde{\alpha}^\top K\tilde{\alpha} + C\sum_i \max\big(0, 1 - y_i f(x_i)\big)$$

where K is the Gram matrix, and $f(x) = f_\alpha(x) = \sum_j \tilde{\alpha}_j k(x, x_j)$.
Thus, in the case of polynomial or radial basis function, the parameters of the SVM model are (d, γ, c). In general, several levels are considered to determine these parameters efficiently.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.3.3 Reproducing kernel Hilbert spaces

In some cases, rather than learning complicated nonlinear classifiers it is better to consider simpler, smoother functions. To do so, we will relate the formulation of the learning problem described in the kernel trick to a more familiar set up, that of Hilbert spaces, in order to understand its applicability.
We have already encounter the kernel function $k(x, \cdot)$ in the case of support vector machines (SVMs). In that setting, the problem can be expressed as

$$f^* = \arg\min_{f \in F} \big(\frac{1}{2}\|f\|^2 + \sum_{i=1}^{N} L(y_i, f(x_i))\big)$$

where $f = f_\alpha = \sum_i \alpha_i k(x_i, \cdot)$. Note, we will need to formally define F to reach our goal.

11.3.3.1 The Hilbert spaces

We let $\mathcal{H}_k$ be a reproducing kernel Hilbert space (RKHS), that is, $\mathcal{H}_k$ is a Hilbert space with some inner product $< \cdot, \cdot >$ and, for the set X, some positive definite kernel function $k : X \times X \to \mathbb{R}$ with the following pair of properties

$$\mathcal{H}_k = \{f : f = \sum_{i=1}^{\infty} \alpha_i k(x_i, \cdot)\}$$

It means that the space consists of all functions resulting from a linear combination of kernel evaluations:

$$< f, k(x_i, \cdot) >= f(x_i)$$

Hence, the kernel functions can be thought of as a kind of basis for the space. More formally, we have:

Definition 11.3.1 *RKHS*
A Hilbert space is called reproducing kernel Hilbert space for kernel function k if both of the following conditions are true:

1. *any function $f \in H$ can be written as an infinite linear combination of kernel evaluations*

$$f = \sum_{i=1}^{\infty} a_i k(x_i, \cdot)$$

   for $x_1, ..., x_m \in X$.

2. *$\mathcal{H}_k$ satisfies the reproducing property*

$$< f, k(x_i, \cdot) >= f(x_i)$$

   That is, the kernel function clamped to one x_i is the evaluating functional for that point.

This definition implies that

$$< k(x_i, \cdot), k(x_j, \cdot) >= k(x_i, x_j) \Leftarrow \text{ entries in the Gram matrix}$$

As a result, if we let F be an RKHS, $F = \mathcal{H}_k$, then we can rewrite the problem (empirical risk function) as

$$f^* = \arg\min_{f \in \mathcal{H}_k} \big(\frac{1}{2}\|f\|^2 + \sum_{i=1}^{N} L(y_i, f(x_i))\big)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

As an example, we consider the square exponential kernel. We let $X \subset \mathbb{R}^N$ and

$$k(x, x^{'}) = e^{\frac{-\|x-x^{'}\|}{h}}$$

Evaluating this kernel at specific points results in Gaussian distributions. More interestingly we can also consider functions that are sums of Gaussian distributions. This is because the superposition of infinitely many Gaussian distributions is a dense set capable of approximating any continuous function.
We can now relate the empirical risk function, described above, to the kernel function as follows:

Theorem 11.3.1 *The representer theorem*
For any data set $\{(x_1, y_1), ..., (x_N, y_N)\}$, $\exists\ \alpha_1, ..., \alpha_N$ *such that* $f^* \in \arg\min\big(\frac{1}{2}\|f\|^2 + \sum_{i=1}^N L(y_i, f(x_i))\big)$ *can be instead written as* $f^* = \sum_{i=1}^N \alpha_i k(x_i, \cdot)$.

11.3.3.2 Nonparametric regression

The representer theorem is used in the case of nonparametric regression (NR): We want to learn some function $f : X \to \mathbb{R}$, but we do not have any prior knowledge about f, so that f can be an arbitrary function. One could use a loss function described above, but when the data contains many points in particular areas with large gaps of little or no data between the clusters, the resulting function could be very different from the target function in those areas. We therefore need to quantify how certain we are of the quality of the fit at various points on the function derived from regression on the data. We want to place confidence intervals around the function to reflect the areas of uncertainty. To do so we need to think in terms of fitting a distribution $P(f)$ over the target function f. We want the properties that low values of $\|f\|$ yield high values of $P(f)$ and vice versa. We can assume that the data (x_i, y_i) is such that $y_i = f(x_i) + \epsilon_i$, which is a function plus some error, the posterior $P(f|D)$. If we have the prior distribution $P(f)$, and the likelihood $P(y|f, x)$ we can compute the posterior distribution $P(f|y, x)$ via the application of Bayes' theorem

$$P(f|y, x) = \frac{P(f)P(y|f, x)}{P(y|x)}$$

We therefore need a suitable prior distribution $P(f)$ and a way to compute the posterior $P(f|D)$ where D is the domain. One solution is to use Gaussian processes as they are associated to the square exponential kernel and their distributions are easy to compute.

11.4 Some discriminative models

In order to make predictions of y for new values of x, we need to model the conditional distribution $P(y|x)$.

11.4.1 Mixture density network

When predicting continuous variables under uncertainty, the conditional expectation provide limited description of the properties of the target variables. One solution is to model the conditional probability distribution of the target data conditioned on the input vector. Bishop [1994] proposed the Mixture Density Network (MDN) obtained by combining a neural network with a mixture density model. The use of the MDN model allows one to do multimodal regression as well as to predict variances.
In this section, we briefly introduce mixture models and then discuss the MDN.

11.4.1.1 Mixture models

Given a vector x of inputs, we wish to predict y by estimating the probability of y given x, $p(y|x)$. If we assume a Gaussian distribution for the real valued target data, we get

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$p(y|\, x;\theta,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(\mu(x,\theta)-y)^2}{2\sigma^2}}$$

where $\mu(x,\theta) = xw + b$, $\theta = \{w, b\}$, is the conditional mean of the Gaussian distribution. That is, we minimise the squared error term $(\mu(x,\theta) - y)^2$ of the output of the linear function μ given x, its parameters θ, and the target value y for all pairs of (x, y) in some dataset D. However, in general, the real valued target data is not Gaussian. One solution is to use mixture models to model arbitrary probability densities.

A mixture model corresponds to the mixture distribution that represents the probability distribution of observations in the overall population (see Marin et al. [2011]). In practice, it is used to make statistical inferences about the properties of the sub-populations given only observations on the pooled population, without sub-population identity information.

We let K be the number of mixture components, N the number of observations, $\theta = \{\theta_1, ..., \theta_L\}$ the parameter of distribution of observation, $\phi = \{\phi_1, ..., \phi_K\}$ the vector of mixture weights with the constraint $\sum_{i=1}^{K} \phi_i = 1$, $x = \{x_1, ..., x_N\}$ the vector of observations, $z = \{z_1, ..., z_N\}$ the vector of component observations, $F(x|\theta)$ the probability distribution of an observation with parameter θ. The conditional probability for a mixture can be represented by a linear combination of kernel functions as

$$\begin{aligned} p(y|x) &= \sum_{i=1}^{K} \phi_i(x)\mathcal{D}_i(y|x) \\ &= \sum_{i=1}^{K} \phi_i(x)\mathcal{D}\big(y|\, \theta_{1,i}(x), ..., \theta_{L,i}(x)\big) \end{aligned}$$

where the weights $\phi_i(x)$ are prior probabilities conditioned on x, $\mathcal{D}$ is the corresponding distribution to be mixed, and θ denotes the parameters of the distribution $\mathcal{D}$.

In a Bayesian setting, the mixture weights and parameters will themselves be random variables, and prior distributions will be placed over the variables. Then, α is a shared hyperparameter for component parameters, β is a shared hyperparameter for mixture weights, $H(\theta|\alpha)$ is the prior probability distribution of component parameters with parameter α. We set $x_{i=1,...,N}|z_{i=1,...,N}, \theta_{i=1,...,K} \sim F(\theta_{z_i})$ where F is an arbitrary distributions over observations. Typically, H is the conjugate prior of F. The two most common choices of F are Gaussian (for real-valued observations) and categorical (for discrete observations).

A Bayesian Gaussian mixture model is commonly extended to fit a vector of unknown parameters, or multivariate normal distributions. This is illustrated in Figure (11.3). In a multivariate distribution with N random variables, we model a vector of parameters (such as several observations of a signal or patches within an image) using a Gaussian mixture model prior distribution on the vector of estimates given by

$$p(\theta) = \sum_{i=1}^{K} \phi_i N(\mu_i, \Sigma_i)$$

where the ith vector component is characterized by normal distributions with weights ϕ_i, means μ_i and covariance matrices Σ_i. To incorporate this prior into a Bayesian estimation, the prior is multiplied with the known distribution $p(x|\theta)$ of the data x conditioned on the parameters θ to be estimated. With this formulation, the posterior distribution $p(\theta|x)$ is also a Gaussian mixture model of the form

$$p(\theta|x) = \sum_{i=1}^{K} \tilde{\phi}_i N(\tilde{\mu}_i, \tilde{\Sigma}_i)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

with new parameters $\tilde{\phi}$, $\tilde{\mu}_i$ and $\tilde{\Sigma}_i$ that are updated using the Expectation-Maximisation (EM) algorithm. Note that this formulation yields a closed-form solution to the complete posterior distribution. Estimations of the random variable θ may be obtained via one of several estimators, such as the mean or maximum of the posterior distribution.

Figure 11.3: Bayesian Gaussian mixture model using plate notation. Smaller squares indicate fixed parameters; larger circles indicate random variables. Filled-in shapes indicate known values.

11.4.1.2 The MDN model

We let $\mathcal{X}$ and $\mathcal{Y}$ be the input and output spaces, $\mathcal{D}$ the probability distribution, D the data set, H the set of hypotheses, $I(\cdot)$ the indicator function. We let $x \in \mathcal{X}$ be an instance, or feature vector, and $y \in \mathcal{Y}$ be the output vector. Here, x takes values in $\mathbb{R}^d$ and y takes values in $\mathbb{R}^c$. We let the network be trained on a finite set of data denoted (x^q, y^q) for $q = 1, ..., n$ and q labels the particular training pattern.
Bishop [1994] let the kernel functions of the conditional distribution be Gaussian of the form

$$\mathcal{D}_i(y|x) = \frac{1}{(2\pi)^{c/2}\sigma_i^c(x)} e^{-\frac{\|y-\mu_i(x)\|^2}{2\sigma_i^2(x)}}$$

where $\mu_i(x)$ is the centre of the ith kernel with components μ_{ik}. He let the parameter of the mixture model be continuous functions of x: the mixing coefficients $\phi_i(x)$, the means $\mu_i(x)$, and the standard deviations $\sigma_i(x)$. They are the output of a neural network taking x as input. This is the Mixture Density Network (MDN) with architecture illustrated in Figure (11.4).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 11.4: MDN architecture.

As an example, the author considered a standard multi-layer perceptron with a single hidden layer of sigmoidal units and an output layer of linear units, and z_j denotes the output variables. The total number of outputs is $(c+2) \times K$. For the mixing parameters to satisfy the unity constraint a softmax function is applied:

$$\phi_i = f_s(z_i^\phi) = \frac{e^{z_i^\phi}}{\sum_{j=1}^K e^{z_j^\phi}}$$

where z_i^ϕ is the corresponding network output. This way ϕ_i lies in the range $(0,1)$ and sum to unity. For the standard deviation to remain positive, the author used an exponential function

$$\sigma_i = e^{z_i^\sigma}$$

The centers μ_i represent location parameters directly represented by the network outputs

$$\mu_{ik} = z_{ik}^\mu$$

Following the same approach as in Section (10.1.4), the likelihood function for the MDN is constructed and the error function is defined by taking the negative logarithm of the likelihood to get the error functions for the MDN in the form

$$E = \sum_q E^q$$

where the contribution to the error from pattern q is given by

$$E^q = -\ln\Big\{\sum_{i=1}^K \phi_i(x^q)\mathcal{D}_i(y^q|x^q)\Big\}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $\mathcal{D}_i(y|x)$ is given above. Here, we model the complete conditional probability density of the output variables. The author minimise the error function by by computing the derivatives of the error E with respect to the weights with standard backpropagation procedure. Since the error function is made of a sum of terms, one for each pattern, $\delta_k^q = \frac{\partial E^q}{\partial z_k}$ is the derivative for the qth pattern. It acts as error which is backpropagated through the network to find the derivatives with respect to the network weights. We can then use an optimisation algorithm to find a minimum of E. Once the MDN is trained, it can be used to predict the conditional density function of the target data for any given value of the input vector. We can then compute more specific quantities such as the conditional average of the target data or the variance of the density function.
To derive the conditional mean from the probability density of the MDN for a single datapoint, one computes:

$$E[y|x] = \int yp(y|x)dy = \phi^\top(x)\mu(x)$$

Note, the mean does not incorporate the standard deviation $\sigma(x)$. Knowing this distribution, we can compute more elaborate quantities such as the Shannon entropy which can serve as an indicator of how certain we are, or the f-divergences to assess similarity between predictions.

11.4.1.3 Summary

In the case of Gaussian mixture model (GMM) with weights ϕ_i, mean μ_i and variance σ_i^2, $i = 1, ...K$, we let the MDN model, $\mathcal{M}$, maps a set of input features x to the parameters of the GMM. The latter returns a full probability density function of an output feature y, conditioned on the input features. That is,

$$p(y|x, \mathcal{M}) = \sum_{i=1}^{K} \phi_i(x) N(y; \mu_i(x), \sigma_i^2(x))$$

The GMM parameters can be derived from the MDN as

$$\begin{aligned}
\phi_i(x) &= \frac{e^{z_i^\phi(x,\mathcal{M})}}{\sum_{j=1}^{K} e^{z_j^\phi(x,\mathcal{M})}} \\
\sigma_i(x) &= e^{z_i^\sigma(x,\mathcal{M})} \\
\mu_i(x) &= z_i^\mu(x, \mathcal{M})
\end{aligned}$$

where $z_i^\phi(x, \mathcal{M})$, $z_i^\sigma(x, \mathcal{M})$, and $z_i^\mu(x, \mathcal{M})$ are the activations of the output layer of the MDN corresponding to the mixture weight, variance, and mean for the m-th Gaussian component in the GMM, given x and $\mathcal{M}$ respectively.
Training of the MDN aims to maximise the log likelihood of $\mathcal{M}$ given the data as

$$\widehat{\mathcal{M}} = \arg\max_{\mathcal{M}} \sum_{n=1}^{N} \sum_{t=1}^{T_{(n)}} \log p\big(y_t^{(n)} \mid x_t^{(n)}, \mathcal{M}\big)$$

where

$$D = \big\{(x_1^{(1)}, y_1^{(1)}), ..., (x_{T_{(1)}}^{(1)}, y_{T_{(1)}}^{(1)}), \cdots, (x_1^{(N)}, y_1^{(N)}), ..., (x_{T_{(N)}}^{(N)}, y_{T_{(N)}}^{(N)})\big\}$$

is the set of input/output pairs in the training data, N is the number of observation in the training data, and $T_{(n)}$ is the number of frames in the n-th training observations.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.4.1.4 Improvements

Nowadays, MDN can be seen as an extension module, applicable to a broad variety of business relevant tasks. The main idea is to combine a deep neural network (DNN) with a mixture of distributions. The DNN provides the parameters for multiple distributions, which are then mixed by some weights. Note, these weights are also provided by the DNN. We obtain the deep MDN (DMDN). The resulting (multimodal) conditional probability distribution can be used to model complex patterns found in real-world data. For instance, Zen et al. [2014] used a mixture density output layer to estimate full probability density functions in statistical parametric speech synthesis (SPSS). As an example, they considered DMDN having three hidden layers with four units per hidden layer, and a mixture density output layer with two Gaussian components. This architecture is illustrated in Figure (11.5).

Figure 11.5: DMDN architecture applied to SPSS.

First, a text to be synthesised is converted to a sequence of linguistic features $\{x_1, ..., x_T\}$. Then, the durations of each speech unit (such as a phoneme) are predicted by a duration prediction module. The probability distributions (GMMs) over acoustic features, including spectral and excitation parameters and their dynamic features given linguistic features, are predicted by a trained DMDN using forward propagation. From the sequence of the predicted GMMs, the authors use a speech parameter generation algorithm to generate smooth trajectories of acoustic features which satisfy the statistics of both static and dynamic features. Finally, a waveform synthesis module outputs a synthesised waveform given the acoustic features.

11.5 Probabilistic topic models

We consider the problem of modelling text corpora, or collections of discrete data. We are interested in finding topics in documents, but we choose not to fix the topics in advance nor to perform manual annotation. This is a clustering problem.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.5.1 Overview

In information retrieval (IR), one reduces each document in the corpus to a vector of real numbers, each of which representing ratios of counts. For instance, the tf-idf scheme reduces documents of arbitrary length to fixed-length list of numbers. However, it does not reveal much of inter or intra document statistical structure. One solution proposed was to apply statistical methods such as Latent Semantic Analysis (LSA) to get some insight (see Landauer et al. [1998]). It assumes that semantic information can be derived from a word document co-occurrence matrix, that dimensionality reduction is an essential part of this derivation, and that words and documents can be represented as points in Euclidean space. The third claim can be alleviated by considering a class of statistical models where the semantic properties of words and documents are expressed in terms of probabilistic topics.
Topic models assume that documents exhibit multiple topics (but typically not many), where a topic is a probability distribution over words (see Hofmann [2001], Griffiths et al. [2002] [2004]). A topic model is a generative model for documents, which specifies a simple probabilistic procedure to generate documents. That is, a document is generated by choosing a distribution over topics, and for each word in the document, we choose randomly a topic and draw a word from that topic. Thus, by statistically inverting this process, we can infer the set of topics responsible for generating a collection of documents. The idea being that each topic provide a probability distribution over words that picks out a coherent cluster of correlated terms, making the topics interpretable.

11.5.1.1 Generative models

As discussed in Section (11.1), topic modelling can be split in two ways: (1) generative model (GM), and (2) a problem of statistical inference (SI).
In document analysis, generative models rely on probabilistic sampling rules for describing how the words in documents are generated on the basis of latent (random) variables. The goal being to find the best possible set of latent variables explaining the observed data (words in documents), assuming that the model is actually generated by the data. This was illustrated by Steyvers et al. [2006]. In Figure (11.6), in the GM (on the left), topics 1 and 2 are related to money and rivers and consist of bags containing different distribution over words. Different documents are generated by picking words from a topic based on a weight assigned to the topic. There is no notion of mutual exclusivity restricting words to be part of one topic only. Further, no assumptions are made on the order of words as they appear in documents. The only relevant information is the number of times words are produced, which is called the bag-of-words assumption. The word-order information is not considered. In the SI (on the right), given the observed words in a set of documents, we want to infer the topic model most likely to have generated the data. That is, infer the probability distribution over words associated with each topic, the distribution over topics for each document, and the topic responsible for generating each word.

Figure 11.6: Generative model and statistical inference.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.5.1.2 Notation

A variety of probabilistic models have been proposed for topic analysis, all assuming that a document is a mixture of topics (see Section (4.3.3.2)), but with different statistical assumptions. Only the number of topics K is specified in advance.
In order to define the joint distribution of the hidden (latent) and observed variables we introduce some notation:

- A word is the basic unit of discrete data, defined to be an item from a vocabulary indexed by $\{1, ..., V\}$. Words are represented by unit-basis vectors having a single component equal to 1 and all other components equal to 0. The ith word in the vocabulary is represented by a V-vector w such that $w_i = 1$ and $w_j = 0$ for $j \neq i$.
- w_d is the V-vector of observed words for document d.
- A document d is a sequence of N_d words denoted by $w_d = (w_{d,1}, ..., w_{d,N_d})$.
- A corpus is a collection of D documents denoted by $C_D = \{w_1, ..., w_D\}$.
- $\beta_{1:K}$ are the topics where each β_k is a distribution over the vocabulary.
- θ_d are the topic proportions (or weights) for document d.
- $\theta_{d,k}$ is the topic proportion (or weight) for topic k in document d.
- z_d are the topic assignments for document d.
- $z_{d,n}$ is the topic assignment for word n in document d.

We write

1. $P(z_d)$ the probability distribution over topics z_d in document d (topic-document distribution), and
2. $P(w_d|\ z_d)$ the probability distribution over words w given topic z_d (word-topic distribution).

Each word $w_{d,i}$ (ith word token) in document d is generated by first sampling a topic from the topic distribution, then choosing a word from the topic-word distribution. We write $P(z_{d,i} = j)$ the probability that the jth topic was sampled from the ith word token, and $P(w_{d,i}|\ z_{d,i} = j)$ the probability of word $w_{d,i}$ under topic j. The model specifies the distribution over words within a document as follows:

$$P(w_{d,i}) = \sum_{j=1}^{K} P(w_{d,i}|\ z_{d,i} = j)P(z_{d,i} = j)$$

To simplify notation, we let

1. $\theta^{(d)} = P(z_d)$ be the multinomial distribution (see Appendix (24.5)) over topics for document d, and
2. $\phi^{(j)} = P(w_d|\ z_d = j)$ be the multinomial distribution over words for topic j.

The parameters ϕ indicate the importance of words for a topic, and θ indicate the importance (or weighting) of topics for a document. The latter is the mixture weights. We also assume that the text collection consists of D documents, and each document d consists of N_d word tokens. N is the total number of word token, $N = \sum_{d=1}^{D} N_d$.

***Remark* 11.5.1** *We omit the subscript d when referring to a single document.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.5.1.3 Geometric interpretation

Hofmann [1999] provided a geometric interpretation to the probabilistic models. Given a vocabulary containing V distinct word types, a V-dimensional space can be constructed where each axis represents the probability of observing a particular word type. The $(V-1)$-dimensional simplex represents all probability distributions over the words. Since a document is a probability distribution over the words, it can be represented as a point on the simplex. Similarly, each topic can be represented as a point on the simplex.
All the latent variable models discussed in this section can be viewed as a point on the $(V-1)$-simplex, called the word simplex. For instance, the unigram model finds a single point on the word simplex and assume that all words in the corpus come from the corresponding distribution. The latent variable models consider K points on the word simplex and form a sub-simplex based on those points. This is called the topic simplex. Any point on the topic simplex is also a point on the word simplex. Each documents generated by the model is a convex combination of the K topics. Thus, all word distributions are points on the $(V-1)$-dimensional simplex, but also points on the $(K-1)$-dimensional simplex spanned by the topics. Note, the different latent variable models use the topic simplex in different ways to generate a document.
The LDA states that each word of both observed and unseen documents is generated by a randomly chosen topic, which is drawn from a distribution with a randomly chosen parameter. The latter is sampled once per document from a smooth distribution on the topic simplex.

11.5.2 Simple latent variable models

Most probabilistic topic models assume that a document is a mixture of topics with mixture weights θ. We introduce a few simple latent variable models. These models can be conveniently illustrated with plate notation. This is a graphical representation where shaded and unshaded variables indicate observed and latent variables, respectively.

11.5.2.1 The unigram model

The words of every document are drawn independently from a single multinomial distribution (see Appendix (24.5)):

$$P(w) = \prod_{n=1}^{N} P(w_n)$$

This is illustrated with plate notation in Figure (11.7).

Figure 11.7: Graphical representation of unigrams.

11.5.2.2 The mixture of unigrams model

We obtain a mixture of unigrams model by augmenting the unigram model with a discrete random topic variable z. Each document is generated by first choosing a topic z and then generating N words independently from the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

conditional multinomial distribution $P(w|\ z)$. The probability of a document is

$$P(w) = \sum_z P(z) \prod_{n=1}^{N} P(w_n|\ z)$$

This is illustrated with plate notation in Figure (11.8).

Figure 11.8: Graphical representation of mixture of unigrams.

Here, $P(z)$ serves as the mixture weights θ. When estimated from a corpus, the word distributions are seen as representations of topics under the assumption that each document exhibits exactly one topic, which is too limiting to effectively model a large collection of documents. With K topics, there are $(K-1)$ parameters associated with $P(z)$ in the mixture of unigrams model.

11.5.2.3 The probabilistic latent semantic indexing model

The probabilistic latent semantic indexing (pLSI) model states that a document d and a word w_n are conditionally independent given an unobserved topic z

$$P(d, w_n) = P(d) \sum_z P(w_n|\ z) P(z|\ d)$$

This is illustrated with plate notation in Figure (11.9).

Figure 11.9: Graphical representation of pLSI.

It captures the fact that a document may contain multiple topics since $P(z|\ d)$ serves as the mixture weights of the topics for a particular document d. Here, d is a dummy index into the list of documents in the training set. Thus, d takes values within the training documents, and the topic mixtures $P(z|\ d)$ are only learned from these trained documents. One can not use to assign probability on an unseen document. Further, the parameters for a K-topic pLSI model are K multinomial distributions of size V and D mixtures over the K hidden topics. It gives $KV + KD$ parameters and linear growth in D, leading to model overfit. In the LDA, the topic mixture weights are treated as K-parameter hidden random variable, so that it can easily generalise to new documents. In addition, the $K + KV$ parameters in LDA do not grow with the size of the training corpus.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.5.3 Latent Dirichlet allocation

Hofmann [2001] introduced the Probabilistic Latent Semantic Indexing method (pLSI), but there is no assumption on the generation of the mixture weights θ. The model was extended by Blei et al. [2003] by introducing a Dirichlet prior on θ, which was called Latent Dirichlet Allocation (LDA). They proposed efficient approximate inference techniques based on variational methods and an EM algorithm for empirical Bayes parameter estimation.

11.5.3.1 The algorithm

Latent Dirichlet allocation (LDA) (see Blei et al. [2003]) is a probabilistic model with a corresponding generative process. It is a three-level hierarchical Baysian model (see Gelman et al. [1995]), where each item (token) of a corpus is modelled as a finite mixture over an underlying set of topics. In turn, each topic is modelled as an infinite mixture over an underlying set of topic probabilities. All documents are assumed to be generated by this process. Thus, a topic is a distribution over a fixed vocabulary. The topics are assumed to be generated before the documents.
The generative process to produce a document d in a corpus is as follows:

1. Randomly choose a distribution over topics. That is, choose the weights $\theta \sim Dir(\alpha)$.
2. For each word w_n in the document
   (a) Randomly choose a topic from the distribution over topics. That is, $z_n \sim Multinom(\theta)$.
   (b) Randomly choose a word w_n from the corresponding topic (distribution over the vocabulary). $P(w_n|\, z_n, \beta)$ is a multinomial probability conditioned on the topic z_n.

where $Dir(\cdot)$ is the Dirichlet distribution of dimension K, and $Multinom(\cdot)$ is the multinomial distribution. Words are generated independently of other words (unigram bag-of-words model). The word probabilities are parametrised by a $K \times V$ matrix β where $\beta_{ij} = P(w_j = 1|\, z_i = 1)$.

11.5.3.2 Description

To simplify the problem, Blei et al. [2003] introduced the Dirichlet distribution as a convenient prior for the mixture weights θ. They let α be a K-vector of parameters of the Dirichlet distribution. In a simplex coordinate system, they considered the weight vector $\theta = (\theta_1, .., \theta_K)$ with $\theta_i \in \{0, ..., n\}$ to satisfy the multinomial distribution. They assumed a unigram language model with 1-of-V coding (V the vocabulary size). θ_i indicates word i of the observed vocabulary, that is,

$$\theta_i = \begin{cases} 1 \text{ word } i \text{ observed} \\ 0 \text{ otherwise} \end{cases}$$

That is, $\theta_i = P(w_i)$ is the probability that word i is seen.
The probability density of a K-dimensional Dirichlet (continuous) distribution with parameters $\alpha = (\alpha_1, ..., \alpha_K)$ over the multinomial distribution for the point $\theta = (\theta_1, ..., \theta_K)$ is given by

$$P(\theta|\, \alpha) = \frac{\Gamma(\sum_{i=1}^K \alpha_i)}{\prod_{i=1}^K \Gamma(\alpha_i)} \prod_{i=1}^K \theta_i^{\alpha_i - 1}$$

where Γ is the Gamma distribution (see Appendix (24.4.1)). The coordinate system is such that the observations satisfy $\sum_{i=1}^K \theta_i = 1$, $\theta_i \geqslant 0$. The Dirichlet is a convenient distribution on the simplex. Each hyperparameter α_i can be interpreted as a prior observation count for the number of times topic i is sampled in a document, before having observed any actual words from that document. Note, one can use a symmetric Dirichlet distribution with a single parameter α such that $\alpha_1 = \cdots = \alpha_K = \alpha$. Hence, placing a Dirichlet prior on the topic distribution θ, we get a smoothed topic distribution with smoothing coefficients α.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Given the parameters α and β, the joint distribution of a topic mixture θ, a set of N topics z, and a set of N words w, is given by

$$P(\theta, z, w|\, \alpha, \beta) = P(\theta|\, \alpha) \prod_{n=1}^{N} P(z_n|\, \theta) P(w_n|\, z_n, \beta) \tag{11.5.13}$$

where $P(z_n|\, \theta)$ is simply θ_i for the unique i such that $z_{n,i} = 1$. Integrating Equation (11.5.13) over θ and summing over z, we get the marginal distribution of a document

$$P(w|\, \alpha, \beta) = \int P(\theta|\, \alpha) \Big(\prod_{n=1}^{N} \sum_{z_n} P(z_n|\, \theta) P(w_n|\, z_n, \beta) \Big) d\theta$$

Taking the product of the marginal probabilities of a single document, the authors obtained the probability of a corpus

$$P(C_D|\, \alpha, \beta) = \prod_{d=1}^{D} \int P(\theta_d|\, \alpha) \Big(\prod_{n=1}^{N_d} \sum_{z_{d,n}} P(z_{d,n}|\, \theta_d) P(w_{d,n}|\, z_{d,n}, \beta) \Big) d\theta_d$$

This model is illustrated with plate notation in Figure (11.10).

Figure 11.10: Graphical representation of LDA.

There are three levels to the LDA representation:

1. The parameters α and β are corpus level parameters, sampled once in the corpus.
2. The parameters θ_d are document-level variables, sampled once per document.
3. The variables $z_{d,n}$ and $w_{d,n}$ are word-level variables, sampled once for each word in each document.

Note, the topic node is sampled repeatedly within the document so that documents can be associated with multiple topics.
In order to use the LDA, one need to compute the posterior distribution of the hidden variables given a document:

$$P(\theta, z|\, w, \alpha, \beta) = \frac{P(\theta, z, w|\, \alpha, \beta)}{P(w|\, \alpha, \beta)} \tag{11.5.14}$$

which is in general intractable. Nonetheless, one can use approximate inference algorithms such as Laplace approximation, variational approximation (see Jordan et al. [1999]), and Markov chain Monte Carlo. Blei et al. [2003] proposed a simple convexity-based variational algorithm for inference in LDA. It use the Jensen's inequality to obtain an adjustable lower bound on the log-likelihood. They consider a family of lower bounds indexed by a set of variational parameters chosen by an optimisation procedure to find the tightest lower bound.
The authors also presented an empirical Bayes method for parameter estimation in the LDA model. Given a corpus of documents C_D, they find the parameters α and β maximising the (marginal) log-likelihood of the data

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$l(\alpha, \beta) = \sum_{d=1}^{D} \log P(w_d | \alpha, \beta)$$

Since $P(w| \alpha, \beta)$ is not tractable, they used the lower bound obtained from variational inference. Thus, they found approximate empirical Bayes estimates for the LDA model via an alternating variational Expectation Maximisation (EM) procedure (see Jordan et al. [1995]).

11.5.3.3 A variant

Griffiths et al. [2002] [2004] considered a variant of this model (also discussed by Blei et al. [2003]) by placing a symmetric $Dirichlet(\beta)$ prior on ϕ. The parameter β is seen as the prior observation count on the number of times words are sampled from a topic before any word from the corpus is observed. It smooths the word distribution in every topic, with the smoothing coefficient β. This approach is illustrated with plate notation in Figure (11.11). The variables ϕ, θ, and z are the three sets of latent variables that we want to infer. Note, the hyperparameters α and β are treated as constants in the model. The arrows indicate conditional dependencies between variables, the plates (boxes in the figure) refer to repetitions of sampling steps with the variable in the lower right corner referring to the number of samples.

Figure 11.11: Graphical representation of LDA.

The joint distribution of the latent and observed variables is given by

$$P(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D}) = \prod_{i=1}^{K} P(\beta_i) \prod_{d=1}^{D} P(\theta_d) \prod_{n=1}^{N} P(z_{d,n}| \theta_d) P(w_{d,n}| \beta_{1:K}, z_{d,n})$$

The main variables of interest are the word-topic distributions ϕ and the topic-document distributions θ for each document. As discussed in the previous section, one common approach is to directly estimate the posterior distribution over z (the assignment of word tokens to topics), given the observed words w, while marginalising out ϕ and θ. Each z_i gives an integer value $[1..K]$ for the topic that word token i is assign to. In the case of millions of word token, the estimation of the posterior over z requires efficient estimation procedures.
Gibbs sampling is a form of Markov Chain Monte Carlo (MCMC) technique. The latter refers to a set of approximate iterative techniques designed to sample values from complex high-dimensional distributions (see Gilks et al. [1996]). Gibbs sampling simulates a high-dimensional distribution by sampling on lower-dimensional subsets of variables,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where each subset is conditioned on the value of all others. The sampling is done sequentially. That is, we sample from each variable one at a time, keeping the current values of the other variables fixed.
Before describing the Gibbs sampling algorithm, the authors introduced the following notation for a collection of documents:

- z_i is the topic assigned to the ith token in the whole collection.
- d_i is the document containing the ith token.
- w_i is the word type of the ith token.
- Z_{-i} is the set of topic assignments of all other tokens.
- $\cdot$ is any remaining information such as the hyperparameters α and β.

The Gibbs sampler considers each word token in the text collection in turn, and estimates the probability of assigning the current word token to each topic, conditioned on the topic assignments to all other word token. From this conditional distribution, a topic is sampled and stored as the new topic assignment for this word token. This conditional distribution is denoted $P(z_i = j|\, Z_{-i}, w_i, d_i, \cdot)$, where $z_i = j$ represents the topic assignment of token i to topic j. It is calculated by

$$P(z_i = j|\, Z_{-i}, w_i, d_i, \cdot) \propto \frac{C^{VK}_{w_i j} + \beta}{\sum_{v=1}^{V} C^{VK}_{vj} + V\beta} \frac{C^{DK}_{d_i j} + \alpha}{\sum_{k=1}^{K} C^{DK}_{d_i k} + K\alpha}$$

where C^{VK} and C^{DK} are matrices of counts, word-topic with size $V \times K$ and document-topic with size $D \times K$, respectively. The vocabulary contains V distinct word types. The matrix C^{VK}_{vj} contains the number of times word v is assigned to topic j, not including the current instance i, and C^{DK}_{dj} contains the number of times topic j is assigned to some word token in document d, not including the current instance i. Note, the above equation gives the unnormalised probability, it must be divided by the sum over all topics k.
The Gibbs sampling algorithm assigns each word token to a random topic in $[1..K]$. For each word token, the count matrices C^{VK} and C^{DK} are first decremented by one for the entries corresponding to the current topic assignment. Then, a new topic is sampled from the above distribution, and the count matrices are incremented with the new topic assignment. Each Gibbs sample is made of the set of topics assignments to all N word tokens in the corpus. In the first stage of the sampling process the samples are discarded as they are poor estimates of the posterior. Afterwards, when they start approximating the target distribution, they are saved at regular space intervals to prevent correlations between samples.
The sampling algorithm provides direct estimates of z for every word. Many applications require estimates $\phi^{'}$ and $\theta^{'}$. The posterior estimates are given by

$$\beta_{ij} = \phi_i^{'(j)} = \frac{C^{VK}_{ij} + \beta}{\sum_{v=1}^{V} C^{VK}_{vj} + V\beta} \text{ and } \theta_{dj} = \theta_j^{'(d)} = \frac{C^{DK}_{dj} + \alpha}{\sum_{k=1}^{K} C^{DK}_{dk} + K\alpha}$$

where β_{ij} is the probability of word type i from topic j, and θ_{dj} is the proportion of topic j in document d.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 12

Dynamical models

12.1 Forecasting models

12.1.1 Regression models

12.1.1.1 Data normalisation

Before training the network, the input data should be normalised [1] This process is usually important for complex machine learning models. Some models even require this procedure before data are fed into the models, since the models internally use distances or feature variances so that without normalisation the results would be heavily affected by the feature with the largest variance or scale. Normalising inputs could also help numerical optimisation method (such as Gradient Descent) converge much faster and more accurately.
There are different ways of normalising the data:

1. Linearly transform all the data to a certain range $[a, b]$, where the minimum of the dataset is mapped to a and the largest value is mapped to b.

2. Calculate the mean and standard deviation of the dataset, subtract each sample by sample mean, and then divide each sample by sample standard deviation.

In case [1], given a time series vector $\mathbf{X} = \{x_1, x_2, ..., x_N\}$, if the desired target range is in the interval $[b_L, b_H]$, with b_L and b_H being lower and upper bonds, then if $m_X = \max(\mathbf{X})$ and $n_X = \min(\mathbf{X})$ solve the equation

$$\begin{aligned} an_X + b &= b_L \\ am_X + b &= b_H \end{aligned}$$

we get the gradient a and intercept b, respectively:

$$\begin{aligned} a &= \frac{b_H - b_L}{m_X - n_X} \\ b &= b_L - an_X = \frac{b_L m_X - b_H n_X}{m_X - n_X} \end{aligned}$$

Thus $Y = aX + b$ will give the mapped data

[1] Data normalisation is the process of transforming raw data into some standard form of data.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\hat{x}_i = (b_H - b_L)\frac{x_i - \min(X)}{\max(X) - \min(X)} + b_L \text{ for } i = 1, ..., N \tag{12.1.1}$$

where $\min(X)$ is the minimal value of all x_i, $\max(X)$ is the maximal value of all x_i, and $\hat{x}_i$ is the normalised value of x_i, respectively. Note, we sometime need to perform operations on the normalised data within a sample window. When comparing the results among different windows we must always convert them back into the original order of magnitude using

$$x_i = [\max(X) - \min(X)]\frac{\hat{x}_i - b_L}{(b_H - b_L)} + \min(X) \text{ for } i = 1, ..., N$$

In case [2], suppose $\mathbf{X}$ is a dataset, with $\bar{\mathbf{X}}$ being sample mean of the dataset and S being the sample standard deviation, then the normalised dataset $\mathbf{X}$ has the following expression:

$$\mathbf{X}' = \frac{\mathbf{X} - \bar{\mathbf{X}}}{S} \tag{12.1.2}$$

Sometimes we use the population mean μ instead of the sample mean $\bar{\mathbf{X}}$. For instance, one may assume the population mean of log return is zero. The choice varies from case to case generally.

In summary, data normalisation is an important part of the model which affects its prediction performance. The goal of normalisation is to make the data more tractable so that when the normalised data is further processed by the model, things become more controllable.

12.1.1.2 Description

In order to model a process, such as market returns, with classic statistical models some statistical properties must be assumed such as stationarity or weak stationarity.
We let $\{X_t; t \in T\}$ be a real-valued random process (r.r.p.) and $\{t_1, ..., t_n\} \subset T$, where $t_1 < t_2 < \cdots < t_n$. For instance, X_t can be the log return of an asset at time t. Further, we assume that $\{X_t\}$ is either serially uncorrelated or with minor lower order serial correlations, but it is dependent. We assume that X_t follows the dynamics

$$X_t = \mu_t + a_t \tag{12.1.3}$$

In finance, a_t is the (stochastic) shock or mean-corrected return (or innovation) of an asset return [2]. The conditional mean and conditional variance of X_t, given the filtration $\mathcal{F}_{t-1}$, are defined by

$$\begin{aligned} \mu_t &= E[X_t|\mathcal{F}_{t-1}] = G(\mathcal{F}_{t-1}) \\ \sigma_t^2 &= Var(X_t|\mathcal{F}_{t-1}) = E[(X_t - \mu_t)^2|\mathcal{F}_{t-1}] = H(\mathcal{F}_{t-1}) \end{aligned}$$

where G and H are well defined functions with $H(\cdot) > 0$. The model for μ_t is the mean equation for X_t, and the model for σ_t^2 is the volatility equation for X_t.

***Remark* 12.1.1** *Some authors use h_t to denote the conditional variance of X_t, in which case the shock becomes $a_t = \sqrt{h_t}\epsilon_t$.*

Both the functions G and H could be non-linear. For simplicity of exposition we first consider the linear case. We will then briefly present some statistical models when H is non-linear. Since we assumed that the serial dependence of a (stock return) series $\{X_t\}$ was weak, if it exists at all, μ_t should be simple and we can assume that X_t follows a

[2] since $a_t = X_t - \mu_t$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

simple time series model such as a stationary $ARMA(p,q)$ (autoregressive moving average) model. In that case, the filtration is given by $\mathcal{F}_{t-1} = \{X_{t-1}, X_{t-2}, ..., X_{t-p}\}$ with p-lags and the drift satisfies

$$\mu_t = \phi_0 + \sum_{i=1}^{p} \phi_i X_{t-i} - \sum_{i=1}^{q} \theta_i a_{t-i} \tag{12.1.4}$$

where ϕ_i and θ_i are real numbers. The function H is constant, that is, $H(\mathcal{F}_{t-1}) = \sigma^2$. The parameters p and q are non-negative integers, and the order (p,q) of the ARMA model may depend on the frequency of the return series.

12.1.1.3 Linear regression

Linear regression (LR) is one of the simplest regression model which is still widely used in industry. This most straight forward method works as a benchmark or criterion when competed against a more complex model. The linear hypothesis for $h_\theta(\mathbf{X}^{(i)})$ is of the form

$$\theta_0 + \sum_{i=1}^{m} \theta_i \mathbf{X}^{(i)}$$

Typically, the root mean square error is calculated as follows:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (h_\theta(\mathbf{X}^{(i)}) - \mathbf{y}^{(i)})^2}$$

For consistency with further discussions about recurrent neural network, $\mathbf{x} = \mathbf{X}^{(i)}$ is set to be the ith column instead of ith row of the design matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$. Then, assuming $\mathbf{y}$ is the target vector, in the sense of normal equation, the solution has the form

$$\hat{\boldsymbol{\theta}} = \mathbf{Y}\mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top)^{-1}$$

In order to mitigate potential overfitting, a regularisation term β is introduced so that it becomes *ridge regression* and the solution turns into

$$\hat{\boldsymbol{\theta}} = \mathbf{Y}\mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top + \beta \mathbf{I})^{-1} \tag{12.1.5}$$

where $\mathbf{I}$ is identity matrix. The following is a pseudo code for training linear regression:

Algorithm 28 LinearRegression Training

```
Function linearTrain U, Y, modelInputs {modelInputs ← N_u, N_y etc.}
m ← nbPointsToIgnore + 1;
S = [1; u(:, m : end)];
D = Y(:, m : end);
W_out = DS^⊤ pinv(SS^⊤ + βI); {Ridge Regression, 'pinv' is pseudo inverse}
EndFunction
```

In the sections presenting some results, the task performance of linear regression is mainly used as a benchmark. If a complex model cannot surpass linear regression, then in terms of efficiency no one would use such complex model and would rather consider the simple linear regression.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

12.1.2 Generalised nonlinear nonparametric models

12.1.2.1 Presentation

Among the various techniques to forecast and classify time series, in particular financial time series, fundamental and technical analysis were the most popular ones. Even though statistical procedures were widely used for pattern recognition, the effectiveness of these methods relies both on model's assumptions and prior knowledge on data properties. To remedy these pitfalls, several classifiers developed, using various data mining and computational intelligence methods such as rule induction, fuzzy rule induction, decision trees, neural networks etc. For instance, the best recognised tools in the currency markets is the artificial neural networks (ANNs), supported by numerous empirical studies (see Ahmed et al. [2010]). Among the different networks existing, the artificial neural networks (ANNs) and the artificial recurrent neural networks (RNNs) are computational models designed by more or less detailed analogy with biological brain modules. The former is presented in Chapter (5) and the latter is introduced in Chapter (6).
As discussed by Barron et al. [1988], neural networks can all be viewed as non-parametric methods for performing nonlinear regressions. This general class of methods is called Learning Networks. Thus, the foremost reason for using ANNs is that there is some nonlinear aspect to the forecasting problem under consideration, taking the form of a complex nonlinear relationship between the independent and dependent variables. The characteristics of financial time series, such as equity stock or currency markets, are influenced by the psychology of traders (behavioural finance) and are strongly nonlinear and hardly predictable (see Maknickiene et al. [2011]). Hence, neural networks are good candidates to learn the financial market.

12.1.2.2 Describing the models

Given a time series $\{x_t, x_{t-1}, ...\}$, we aim to learn from the known data to predict future values. That is, we want to estimate x at some future time

$$\widehat{x}_{t+h} = f(x_t, x_{t-1}, ..., x_{t-p}, z_1, z_2, .., z_q)$$

where h is the horizon prediction, and z_q is the qth other explanatory variable. Before training the network, the input data should be normalised into the interval $[b_L, b_H]$, with b_L and b_H being lower and upper bonds (see Section (12.1.1.1)). As discussed in Section (12.1.1), the statistical approach to forecasting time series involves the construction of stochastic models to predict the value $\widehat{x}_{t+h}$ given previous observations. One approach is to use an $ARMA(p,q)$ model, but it lacks the ability to capture the nonlinear features of financial time series. Another approach is to extend the linear models to nonlinear ones. For example, we can generalise an $AR(p)$ model to become a *nonlinear autoregressive* (NAR) model (see Connor et al. [1994])

$$x_t = h(x_{t-1}, x_{t-2}, \cdots, x_{t-p}) + e_t, \tag{12.1.6}$$

where h is an unknown smooth function. Assuming that $E(e_t|x_{t-1}, x_{t-2}, \cdots) = 0$ and e_t has finite second moment, then the minimum mean square error optimal predictor of x_t with the knowledge of $x_{t-1}, ..., x_{t-p}$ is the conditional mean:

$$\widehat{x}_t = E(x_t|x_{t-1}, \cdots, x_{t-p}) = h(x_{t-1}, \cdots, x_{t-p}), t > p$$

Alternatively, as discussed in Section (10.1), we can use neural networks (NNs) (see Chapter (5)) to approximate the function and make some predictions.
Various authors showed that neural networks can be derived from the classical regularisation problem where some unknown function

$$y = h(x)$$

can be approximated from the dataset $\{(x_t, y_t)\}$ and some smoothness constraints. In terms of regression analysis (see Section (12.1.1)), the function h is the conditional expectation of y_t given x_t. For instance, Lapedes [1987] showed

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

that feedforward networks (see Section (5.1.3)) were NAR models for time series prediction. Poggio et al. [1990] followed a similar approach in the case of radial basis function (RBF).
For simplicity of exposition we consider the problem of mapping multiple input variables to a single output variable, like in regression analysis. The predictor is given by the approximation of h as follow

$$\widehat{x}_t = \widehat{h}(x_{t-1}, \cdots, x_{t-p})$$

with

$$\widehat{h}(x_{t-1}, \cdots, x_{t-p}) = \sum_{i=1}^{N_h} w_i f(\sum_{j=1}^{p} w_{ij} x_{t-j} + b_i) \tag{12.1.7}$$

A simple way to visualise the structure of the network is to graph the connections between inputs, nonlinear hidden units and outputs. Figure 12.1 shows the general architecture of the NAR as an extension of the $AR(p)$ in a feedforward network. The weights, $\{w_i\}$ and $\{w_{ij}\}$ in Equation (12.1.7) can be seen as knobs defining the input-output function, $\widehat{h}$, of the network. N_h is the number of neurons in the second layer, f is a monotonic, smooth, and bounded function, b_i is a bias. The weights of the network, $\{w_i\}$ and $\{w_{ij}\}$ are estimated through supervised learning process given a time series sample $x_0, x_1, \cdots, x_n$, and the input-target pairs in the training set are $((x_{t-1}, x_{t-2}, \cdots, x_{t-p}), x_t)$, where $p > 0, t \geqslant p$.

Figure 12.1: Feedforward network for nonlinear autoregressive model.

More generally, to illustrate the nonlinearity of ANNs, we consider a multilayer network (see Section (5.2)) with $p = 1, .., P$ input-output training pairs where $p = P$ is the most recent observation, and assume a set of artificial neurons $\{f_{i,j,l}\}_{k=0}^{K}$ where the multilayer subscript $k = 0$ corresponds to the set of inputs $\{x_i\}_{i=1}^{N_0}$ and $k = K$ corresponds to the set of outputs $\{y_i\}_{i=1}^{N_K}$. For simplicity of exposition, we focus on a system with one hidden layer where the subscripts i and j represent the nodes on the input ($(K-2)$ layer) and the $(K-1)$th hidden layer, and the subscript s represent the nodes on the kth layer, respectively. In that setting, the output $O_{s,K}^p(x, w)$ can be expressed as

$$O_{s,K}^p(x, w) = g_{N_K}\Big(\sum_{j=1}^{N_{K-1}} g_{N_{K-1}}\big(\sum_{i=1}^{N_{K-2}} O_{i,K-2}^p(x, w) w_{i,j}^{k-2} + b_{j,K-1}\big) w_{j,s}^{k-1} + b_{s,K}\Big), s = 1, .., N_K \tag{12.1.8}$$

where $O_{i,K-2}^p(x, w) = x_{p,i}$ is the ith element of the vector input, and $g_{N_K}(\cdot)$ and $g_{N_{K-1}}(\cdot)$ are two possibly different functions. In the special case where $g_{N_K}(\cdot)$ is a linear function and $g_{N_{K-1}}(\cdot)$ is a sigmoid function, the system simplifies to

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$O^p_{s,K}(x,w) = \sum_{j=1}^{N_{K-1}} g_{n_{K-1}}\Big(\sum_{i=1}^{N_{K-2}} O^p_{i,K-2}(x,w)w^{k-2}_{i,j} + b_{j,K-1}\Big)w^{k-1}_{j,s} + b_{s,K}\ , s = 1,..,N_K$$

which can be compared to the ordinary least squares (OLS) regression

$$y^p_s = \alpha + \sum_{i=1}^{N_{K-2}} \beta_i x_{p,i} + \epsilon_s\ , s = 1,..,N_K$$

Hence, we see that the OLS regressor variables y^p_s are put through a transformation, or squashing function, given by $g_{N_{K-1}}(\cdot)$. By iterating within sample and examining the error terms, the neural network modify the input weights $\{w^{k-2}_{i,j}\}$ and the output weights $\{w^{k-1}_{j,s}\}$ to minimise within-sample measure for point prediction models. It can also maximise the classification rate of correct predictions for classification models.
It is not difficult to see that Equation (12.1.8) corresponds to the rate dynamics in Equation (12.1.3) with nonlinear G function. That is,

$$y = g_K\Big(a + \sum_{i=1}^{p} \phi_i x_i + \sum_{j=1}^{q} \theta_j g_{K-1}\big(\alpha_j + \sum_{i\to j} \omega_{ij} x_i\big)\Big)$$

where $x_1, ..., x_p$ are input values, y is the output, for the q-hidden nodes. If the activation function g_K is linear, the feedforward neural network (FFNN) becomes an autoregressive moving average model with nonlinear moving average part. Hence, the FFNN (with one hidden layer) is a generalisation of $ARMA(p,q)$ models.

12.1.3 Example: forecasting volatility

12.1.3.1 Presentation

When visualising financial time series, one can observe heteroskedasticity [3], with periods of high volatility and periods of low volatility, corresponding to periods of high and low risks, respectively. We also observe returns having very high absolute value compared with their mean, suggesting fat tail distribution for returns, with large events having a larger probability to appear when compared to returns drawn from a Gaussian distribution. Hence, besides the return series, we must also consider the volatility process and the behaviour of extreme returns of an asset (the large positive or negative returns). The negative extreme returns are important in risk management, whereas positive extreme returns are critical to holding a short position. Volatility is important in risk management as it provides a simple approach to calculating the value at risk (VaR) of a financial position. Further, modelling the volatility of a time series can improve the efficiency in parameter estimation and the accuracy in interval forecast. As returns may vary substantially over time and appear in clusters, the volatility process is concerned with the evolution of conditional variance of the return over time. Users must make sure that volatilities and correlations are predictable and that their forecasts incorporate the most useful information available. As the forecasts are based on historical data, the estimators must be flexible enough to account for changing market conditions. One simple approach is to assume that returns are governed by the random walk model, and that the sample standard deviation $\hat{\sigma}_N$ or the sample variance $\hat{\sigma}^2_N$ of returns for N periods of time can be used as a simple forecast of volatility of returns, r_t, over the future period $[t+1, t+h]$ for some positive integer h. However, volatility has some specific characteristics such as

- volatility clusters: volatility may be high for certain time periods and low for other periods.
- continuity: volatility jumps are rare.
- mean-reversion: volatility does not diverge to infinity, it varies within some fixed range so that it is often stationary.

[3] Heteroskedastic means that a time series has a non-constant variance through time.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- volatility reacts differently to a big price increase or a big price drop.

These properties play an important role in the development of volatility models. As a result, there is a large literature on econometric models available for modelling the volatility of an asset return, called the conditional heteroscedastic (CH) models. Some univariate volatility models include the autoregressive conditional heteroscedastic (ARCH) model of Engle [1982], the generalised ARCH (GARCH) model of Bollerslev [1986], the exponential GARCH (EGARCH) of Nelson [1991], the stochastic volatility (SV) models and many more. Tsay [2002] discussed the advantages and weaknesses of each volatility model and showed some applications of the models. Unfortunately, stock volatility is not directly observable from returns as in the case of daily volatility where there is only one observation in a trading day. Even though one can use intraday data to estimate daily volatility, accuracy is difficult to obtain. The unobservability of volatility makes it difficult to evaluate the forecasting performance of CH models and heuristics must be developed to estimate volatility on small samples.

12.1.3.2 A first decomposition

As risk is mainly given by the probability of large negative returns in the forthcoming period, risk evaluation is closely related to time series forecasts. The desired quantity is a forecast for the probability distribution (pdf) $\tilde{p}(r)$ of the possible returns r over the risk horizon ΔT. This problem is generally decomposed into forecasts for the mean and variance of the return probability distribution

$$r_{\Delta T} = \mu_{\Delta T} + a_{\Delta T}$$

with

$$a_{\Delta T} = \sigma_{\Delta T}\epsilon$$

where the return $r_{\Delta T}$ over the period ΔT is a random variable, $\mu_{\Delta T}$ is the forecast for the mean return, and $\sigma_{\Delta t}$ is the volatility forecast. The term $a_{\Delta T} = r_{\Delta T} - \mu_{\Delta T}$ is the mean-corrected asset return. The residual ϵ, which corresponds to the unpredictable part, is a random variable distributed according to a pdf $p_{\Delta T}(\epsilon)$. The standard assumption is to let $\epsilon(t)$ be an independent and identically distributed (iid) random variable. In general, a risk methodology will set the mean μ to zero and concentrate on σ and $p(\epsilon)$. To validate the methodology, we set

$$\epsilon = \frac{r - \mu}{\sigma}$$

compute the right hand side on historical data, and obtain a time series for the residual. We can then check that ϵ is independent and distributed according to $p(\epsilon)$. For instance, we can test that ϵ is uncorrelated, and that given a risk threshold α (say, 95%), the number of exceedance behaves as expected. However, when the horizon period ΔT increases, it becomes very difficult to perform back testing due to the lack of data. Alternatively, we can consider a process to model the returns with a time increment δt of one day, computing the forecasts using conditional averages. We can then relate daily data with forecasts at any time horizon, and the forecasts depend only on the process parameters, which are independent of ΔT and are consistent across risk horizon. The quality of the volatility forecasts is the major determinant factor for a risk methodology. The residuals can then be computed and their properties studied.

12.1.3.3 The structure of volatility models

12.1.3.3.1 Presentation The above heuristics being poor estimates of the future volatility, one must rely on proper volatility models such as the conditional heteroscedastic (CH) models. Since the early 80s, volatility clustering spawned a large literaturure on a new class of stochastic processes capturing the dependency of second moments in a phenomenological way. As the lognormal assumption is not consistent with all the properties of historical stock returns, Engle [1982] first introduced the autoregressive conditional heteroscedasticity model (ARCH) which has been generalised to GARCH by Bollerslev [1986].

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We consider the log return r_t to follow the $ARMA(p,q)$ model described in the above Appendix. The excess kurtosis values, measuring deviation from the normality of the returns, are indicative of the long-tailed nature of the process. Hence, one can then compute and plot the autocorrelation functions for the returns process r_t as well as the autocorrelation functions for the squared returns r_t^2.

1. If the securities exhibit a significant positive autocorrelation at lag one and higher lags as well, then large (small) returns tend to be followed by large (small) returns of the same sign. That is, there are trends in the return series. This is evidence against the weakly efficient market hypothesis which asserts that all historical information is fully reflected in prices, implying that historical prices contain no information that could be used to earn a trading profit above that which could be attained with a naive buy-and-hold strategy which implies further that returns should be uncorrelated. In this case, the autocorrelation function would suggest that an autoregressive model should capture much of the behaviour of the returns.

2. The autocorrelation in the squared returns process would suggest that large (small) absolute returns tend to follow each other. That is, large (small) returns are followed by large (small) returns of unpredictable sign. It implies that the returns series exhibits volatility clustering where large (small) returns form clusters throughout the series. As a result, the variance of a return conditioned on past returns is a monotonic function of the past returns, and hence the conditional variance is heteroskedastic and should be properly modelled.

The conditional heteroscedastic (CH) models are capable of dealing with this conditional heteroskedasticity. The variance in the model described in Equation (12.1.3) becomes

$$\sigma_t^2 = Var(r_t|\mathcal{F}_{t-1}) = Var(a_t|\mathcal{F}_{t-1})$$

Since the way in which σ_t^2 evolves over time differentiate one volatility model from another, the CH models are concerned with the evolution of the volatility. Hence, modelling conditional heteroscdasticity (CH) amounts to augmenting a dynamic equation to a time series model to govern the time evolution of the conditional variance of the shock. We distinguish two types or groups of CH models, the first one using an exact function to govern the evolution of σ_t^2, and the second one using a stochastic equation to describe σ_t^2. For instance, the (G)ARCH model belongs to the former, and the stochastic volatility (SV) model belongs to the latter. In general, we estimate the conditional mean and variance equations jointly in empirical studies.

12.1.3.3.2 Benchmark volatility models The ARCH model, which is the first model providing a systematic framework for volatility modelling, states that

1. the mean-corrected asset return a_t is serially uncorrelated, but dependent

2. the dependence of a_t can be described by a simple quadratic function of its lagged values.

Specifically, setting $\mu_t = 0$ for simplicity, an $ARCH(p)$ model assumes that

$$r_t = h_t^{\frac{1}{2}}\epsilon_t \text{ , } h_t = \alpha_0 + \alpha_1 r_{t-1}^2 + ... + \alpha_p r_{t-p}^2$$

where $\{\epsilon_t\}$ is a sequence of i.i.d. random variables with mean zero and variance 1, $\alpha_0 > 0$, and $\alpha_i \geqslant 0$ for $i > 0$. In practice, ϵ_t follows the standard normal or a standardised Student-t distribution. Generalising the ARCH model, the main idea behind (G)ARCH models is to consider asset returns as a mixture of normal distributions with the current variance being driven by a deterministic difference equation

$$r_t = h_t^{\frac{1}{2}}\epsilon_t \text{ with } \epsilon_t \sim N(0,1) \tag{12.1.9}$$

and

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$h_t = \alpha_0 + \sum_{i=1}^{p} \alpha_i r_{t-i}^2 + \sum_{j=1}^{q} \beta_j h_{t-j} \, , \alpha_0 > 0 \, , \alpha_i, \beta_j > 0 \tag{12.1.10}$$

where $\alpha_0 > 0$, $\alpha_i \geqslant 0$, $\beta_j \geqslant 0$, and $\sum_{i=1}^{\max(p,q)} (\alpha_i + \beta_i) < 1$. The latter constraint on $\alpha_i + \beta_i$ implies that the unconditional variance of r_t is finite, whereas its conditional variance h_t evolves over time. In general, empirical applications find the $GARCH(1,1)$ model with $p = q = 1$ to be sufficient to model financial time series

$$h_t = \alpha_0 + \alpha_1 r_{t-1}^2 + \beta_1 h_{t-1} \, , \alpha_0 > 0 \, , \alpha_1, \beta_1 > 0 \tag{12.1.11}$$

When estimated, the sum of the parameters $\alpha_1 + \beta_1$ turns out to be close to the non-stationary case, that is, mostly only a constraint on the parameters prevents them from exceeding 1 in their sum, which would lead to non-stationary behaviour. Different extensions of GARCH were developed in the literature with the objective of better capturing the financial stylised facts. Among them are the Exponential GARCH (EGARCH) model proposed by Nelson [1991] accounting for asymmetric behaviour of returns, the Threshold GARCH (TGARCH) model of Rabemananjara et al. [1993] taking into account the leverage effects, the regime switching GARCH (RS-GARCH) developed by Cai [1994], and the Integrated GARCH (IGARCH) introduced by Engle et al allowing for capturing high persistence observed in returns time series. Nelson [1990] showed that Ito diffusion or jump-diffusion processes could be obtained as a continuous time limit of discrete GARCH sequences. In order to capture stochastic shocks to the variance process, Taylor [1986] introduced the class of stochastic volatility (SV) models whose instantaneous variance is driven by

$$\ln(h_t) = k + \phi \ln(h_{t-1}) + \tau \xi_t \text{ with } \xi_t \sim N(0,1) \tag{12.1.12}$$

This approach has been refined and extended in many ways. The SV process is more flexible than the GARCH model, providing more mixing due to the co-existence of shocks to volatility and return innovations. However, one drawback of the GARCH models and extension to Equation (12.1.12) is their implied exponential decay of the autocorrelations of measures of volatility which is in contrast to the very long autocorrelation. Both the GARCH and the baseline SV model are only characterised by short-term rather than long-term dependence. In order to capture long memory effects, the GARCH and SV models were expanded by allowing for an infinite number of lagged volatility terms instead of the limited number of lags present in Equations (12.1.10) and (12.1.12). To obtain a compact characterisation of the long memory feature, a fractional differencing operator was used in both extensions, leading to the fractionally integrated GARCH (FIGARCH) model introduced by Baillie et al. [1996], and the long-memory stochastic volatility model of Breidt et al. [1998]. As an intermediate approach, Dacorogna et al. [1998] proposed the heterogeneous ARCH (HARCH) model, considering returns at different time aggregation levels as determinants of the dynamic law governing current volatility. In this model, we need to replace Equations (12.1.10) with

$$h_t = c_0 + \sum_{j=1}^{n} c_j r_{t,t-\Delta t_y}^2$$

where $r_{t,t-\Delta t_y} = \ln(p_t) - \ln(p_{t-\Delta t_j})$ are returns computed over different frequencies. This model was motivated by the finding that volatility on fine time scales can be explained to a larger extend by coarse-grained volatility than vice versa (see Muller et al. [1997]). Thus, the right hand side of the above equation covers local volatility at various lower frequencies than the time step of the underlying data $(\Delta t_j = 2, 3, ..)$. Note, multifractal models have a closely related structure but model the hierarchy of volatility components in a multiplicative rather than additive format.

12.2 Introduction to HMM

As discussed in Section (11.2), statistical models such as state-space models (SSMs) include hidden Markov models (HMMs) (see Baum et al. [1966] [1970]) and linear Gaussian SSMs or the Kalman filtering model (see Kalman [1960]). SSMs are a class of probabilistic graphical models that encode the dependency structure of sequential data

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

by latent variables that follow Markovian dynamics. HMM was originally implemented for speech processing applications (see Jelinkek [1969], Baker [1975]). HMMs are known for their applications to thermodynamics, statistical mechanics, physics, chemistry, economics, finance, signal processing, information theory, pattern recognition, such as speech, handwriting, gesture recognition, part-of-speech tagging, musical score following, partial discharges and bioinformatics.
In this section, we are going to introduce hidden Markov models. See textbooks by Murphy [2012] and lecture notes by Rabiner et al. [1986], Ghahramani [2001], Degirmenci [2014].

12.2.1 Description

12.2.1.1 The model

Hidden Markov model (HMM) is a (parameterised) tool for representing probability distributions over sequences of observations. It is a Bayesian finite state process, with a Markovian prior for modelling the transitions between the states, and a set of state probability density functions for modelling the random variations of the signal process within each state. It can be used to model a time-varying random process as a probabilistic Markovian chain of N stationary, or quasi-stationary, elementary sub-processes.
We let X_t be an observation at time t produced by a stochastic process with unobservable (or hidden) state Z_t. It can be a symbol from a discrete alphabet, a real-valued variable, an integer, among others, as long as it can be used to define a probability distribution. The hidden process is assumed to satisfy the Markov property, where the state Z_t at time t only depends on the previous state Z_{t-1} at time $t-1$. This is illustrated in Figure (12.2). An nth order Markov model depends on the n-previous state. Hence, the HMM is a first-order Markov model.

Figure 12.2: Computational graph of a HMM. The hidden states Z are shaded in gray.

In the HMM, the observations $X_{1:N} = X_1, ..., X_N$ come from a known, finite set called the observation space denoted $\mathcal{X}$, and the hidden states $Z_{1:N} = Z_1, ..., Z_N$ come from a known, finite set called the hidden state space denoted $\mathcal{Z}$. Regarding notation, we sometime replace N with T, and vice versa.
Using the chain rule, the probability to any given sequence $Z_{1:N}$ is given by

$$P(Z_{1:N}) = \prod_{t=1}^{N} P(Z_t|\, Z_0, Z_1, ..., Z_{t-1}) = \prod_{t=1}^{N} P(Z_t|\, Z_{<t})$$

where Z_0 is a starting state, and $Z_{<t}$ denotes $Z_1, ..., Z_{t-1}$. Further, the Markov property is assumed

$$P(Z_t|\, Z_{<t}) = P(Z_t|\, Z_{t-1})$$

Hence, the above probability of a sequence simplifies to

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$P(Z_{1:N}) = \prod_{t=1}^{N} P(Z_t|\ Z_{t-1})$$

At last, the independence assumption states that each observation X_t only depends on the immediate hidden state Z_t, that is,

$$P(X_t|\ X_{<t}, Z_{<t}) = P(X_t|\ Z_{<t})$$

Thus, under the Markov assumption and the independence assumption, the HMM only needs two set of parameters to model the transition probability $P(Z_t|\ Z_{t-1})$ and the emission (or observation) probability $P(X_t|Z_t)$, denoted A and B, respectively. Note, the transition probability $P(Z_t|\ Z_{t-1})$ can be decomposed into a deterministic component and a stochastic one as follows:

$$Z_t = f_t(Z_{t-1}) + \epsilon_{Z,t}$$

where f_t is a deterministic transition function, and similarly for the observation probability $P(X_t|Z_t)$

$$X_t = g_t(Z_{t-1}) + \epsilon_{X,t}$$

where g_t is a deterministic output function. When both f_t and g_t are linear and time-invariant, and the noises $\epsilon_{Z,t}$ and $\epsilon_{X,t}$ are Gaussian, we recover a linear-Gaussian state-space model.
The joint distribution of a sequence of states and observation is given by

$$P(Z_{1:N}, X_{1:N}) = P(Z_1)P(X_1|\ Z_1)\prod_{t=2}^{N} P(Z_z|\ Z_{t-1})P(X_t|\ Z_t)$$

Note, we can also write the joint distribution as

$$P(Z_{1:N}, X_{1:N}) = P(Z_1)\prod_{t=2}^{N} P(Z_z|\ Z_{t-1})\prod_{t=1}^{N} P(X_t|\ Z_t)$$

Taking the logarithm of the joint distribution, we get

$$\log P(Z_{1:N}, X_{1:N}) = \log P(Z_1) + \sum_{t=2}^{N} \log P(Z_z|\ Z_{t-1}) + \sum_{t=1}^{N} \log P(X_t|\ Z_t)$$

Note, since the state variables are hidden, we cannot directly compute this equation. We can go around by using the Baum-Welch algorithm, which allows us by computing the expectation of this equation under the posterior distribution of the hidden states given the observations.
HMM is characterised by

1. The number of states K in the model: this is the number of states within the hidden Markov process. That is, Z_t can take on K values denoted by the integers $\{1, ..., K\}$.

2. The number of distinct observations M: the observations are represented as integers $1, ..., M$. The observations X_t at time t are encoded into a $M \times 1$ vector of binary elements, where the only non-zero element is the lth element ($X_{t,l} = 1$) corresponding to state $l \in M$ at time t.

3. The state transition matrix A: It is a $K \times K$ matrix whose elements A_{ij} describe the transition probability from state $Z_{t-1,i}$ to state $Z_{t,j}$ in one time step where $i, j \in \{1, ..., K\}$. It is written by

$$A_{ij} = P(Z_{t,j} = 1|\ Z_{t-1,i} = 1)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Each row of A sum to 1, that is, $\sum_j A_{ij} = 1$, so that we get a stochastic matrix. If any state can reach any other state in a single step (fully connected), then $A_{i,j} > 0$ for all i, j, otherwise A will have some zero-valued elements. The conditional probability can be written as

$$P(Z_t|\, Z_{t-1}) = \prod_{i=1}^{K}\prod_{j=1}^{K} A_{ij}^{Z_{t-1,i}Z_{t,j}}$$

Taking the logarithm, we get

$$\begin{aligned}\log P(Z_t|\, Z_{t-1}) &= \sum_{i=1}^{K}\sum_{j=1}^{K} Z_{t-1,i}Z_{t,j}\log A_{ij}\\ &= Z_t^\top(\log A)Z_{t-1}\end{aligned}$$

using matrix notation.
Note, HMM can be augmented to allow for input variables U_t, such that there is an input dependent state transition probability $P(Z_t|\, Z_{t-1}, U_t)$.

4. The emission probability B (or output probability): It is an $M \times K$ matrix whose elements B_{kj} describe the probability of making observation $X_{t,k}$ given state $Z_{t,j}$. It is written as

$$B_{kj} = P(X_t = k|\, Z_t = j)$$

The conditional probability can be written as

$$P(X_t|\, Z_t) = \prod_{j=1}^{K}\prod_{k=1}^{M} B_{kj}^{Z_{t,j}Z_{t,k}}$$

Taking the logarithm, we get

$$\begin{aligned}\log P(X_t|\, Z_t) &= \sum_{j=1}^{K}\sum_{k=1}^{M} Z_{t,j}X_{t,k}\log A_{kj}\\ &= X_t^\top(\log B)Z_t\end{aligned}$$

using matrix notation.
Note, for real-valued observation vectors, the output model $P(X_t|\, Z_t)$ can take several forms, such as a Gaussian, mixture of Gaussian, or neural network. In the case of high-dimensional real-valued observations, one can use a factor analyser (FA) (closely related to principal components analysis).

5. The initial state distribution π: It is a $K \times 1$ vector of probabilities $\pi_i = P(Z_{1,i} = 1)$. The conditional probability can be written as

$$P(Z_1|\, \pi) = \prod_{i=1}^{K} \pi_i^{Z_{1,i}}$$

These parameters fully characterise the HMM. It is denoted by $\lambda = (A, B, \pi)$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

12.2.1.2 Maximum likelihood revisited

We briefly describe the maximum likelihood (ML) of a network with complete data and in presence of hidden variables.

12.2.1.2.1 Complete data Given a data set of i.i.d observations $D = \{X_1, ..., X_N\}$, where X_i can be a vector or time series of vectors. Given a model M (such as a Bayesian network), the likelihood of the dataset is a function of the parameters of the model

$$P(D|\,\theta, M) = \prod_{i=1}^{N} P(X_i|\,\theta, M)$$

For ease of notation, we drop the implicit conditioning to the model structure. The ML parameters are obtained by maximising the log-likelihood

$$\mathcal{L}(\theta) = \prod_{i=1}^{N} \log P(X_i|\,\theta)$$

In the case where the observation vector includes all the variables in the network, then each term in the log-likelihood further factors as follows:

$$\begin{aligned} \log P(X_i|\,\theta) &= \log \prod_j P(X_i|\,X_{i,pa(j)}, \theta_j) \\ &= \sum_j \log P(X_i|\,X_{i,pa(j)}, \theta_j) \end{aligned}$$

where j indexes the nodes in the network, $pa(j)$ is the set of parents of j, and θ_j are the parameters defining the conditional probability of Y_j given its parents. Thus, the likelihood simplifies the estimation problem by splitting the products into local terms involving each node and its parents.

12.2.1.2.2 Estimation with hidden variables In the presence of hidden variables, we can no-longer compute the log-likelihood as above. In that case, the likelihood for a single observation is

$$\mathcal{L}(\theta) = \log P(X|\,\theta) = \log \sum_Z P(X, Z|\,\theta)$$

where Z is the set of hidden variables, and $P(X|\,\theta)$ is the marginal probability of the data (or likelihood function). It is difficult to directly maximise $\mathcal{L}(\theta)$ with respect to θ since the logarithm of the sum can couple all of the parameters of the model. The problem can be simplified by making use of the following insight: any distribution $Q(Z)$ over the hidden variables defines a lower bound on $\mathcal{L}$. That is,

$$\begin{aligned} \log \sum_Z P(X, Z|\,\theta) &= \log \sum_Z Q(Z) \frac{P(X, Z|\theta)}{Q(Z)} \\ &\leqslant \sum_Z Q(Z) \log \frac{P(Z, X|\theta)}{Q(Z)} \\ &\leqslant \sum_Z Q(Z) \log P(Z, X|\theta) - \sum_Z Q(Z) \log Q(Z) \\ &= \mathcal{F}(Q, \theta) \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where we have use the Jensen's inequality since the logarithm function is concave. If we let $\log P(Z, X|\, \theta)$ be the energy of a global configuration (Z, X), then the lower bound $\mathcal{F}(Q, \theta) \leqslant \mathcal{L}(\theta)$ is the negative of the free energy: the expected energy under Q minus the entropy of Q (statistical result).
The expectation-Maximisation (EM) algorithm (see Baum et al. [1970], Dempster et al. [1977]) alternates between maximising $\mathcal{F}$ with respect to Q and θ, respectively, holding the other one fixed. That is, starting from some initial parameters θ_0, we get

1. E-step: $Q_{k+1} \leftarrow \arg\max_Q \mathcal{F}(Q, \theta_k)$
2. M-step: $\theta_{k+1} \leftarrow \arg\max_\theta \mathcal{F}(Q_{k+1}, \theta)$

The maximum in step [1] is obtained by setting $Q_{k+1}(Z) = P(Z|\, X, \theta_k)$, at which point the bound becomes an equality, $\mathcal{F}(Q_{k+1}, \theta_k) = \mathcal{L}(\theta_k)$. The maximum in step [2] is obtained by maximising the first term (in the above derivation of the lower bound), since the entropy of Q does not depend on θ

$$\text{M step}: \theta_{k+1} \leftarrow \arg\max_\theta P(Z|\, X, \theta_k) \log P(Z, X|\, \theta)$$

Note, EM can be interpreted as coordinate ascent in $\mathcal{F}$: since $\mathcal{F} = \mathcal{L}$ at the beginning of each step [1], and since step [2] does not modify θ, we are guaranteed not to decrease the likelihood after each combined EM step. In addition, we do not need to compute explicitly the posterior distribution $P(Z|\, X, \theta_k)$. Since $\log P(Z, X|\, \theta)$ contains both hidden and observed variables in the network, it can be factored as the sum of logarithm probabilities of each node given its parents. Thus, the quantities required for step [2] are the expected values, under the posterior distribution $P(Z|mboxX, \theta_k)$, of the analogous sufficient statistics required for ML estimation in the complete data case.

12.2.2 The forward-backward algorithm

As discussed in Section (11.2.5.2), the forward-backward algorithm is a dynamic programming algorithm makes use of message passing to compute the filtered marginal $P(Z_t|X_{1:T})$ and the smoothed marginal (with ψ_t being replaced with $X_{1:t}$). We follow the presentation given by Murphy [2012].

12.2.2.1 The forward algorithm

We start by describing the forward algorithm which compute the filtered marginal $P(Z_t|X_{1:T})$ by using the predict-update cycle. Note, Rabiner et al. [1986] and Rabiner [1989] defined the forward variable $\alpha_t(i) = P(X_{1:t}, Z_t = i|\, \lambda)$ as the joint probability of the partial observation sequence $X_{1:t}$ until time t and the state $Z_t = i$ at time t. The forward pass compute α_t recursively as follows:

$$\begin{aligned}
\alpha_t &= P(Z_t, X_{1:t}) \\
&= \Big[\sum_{Z_{t-1}} P(Z_{t-1}, X_{1:t-1}) P(Z_t|\, Z_{t-1})\Big] P(X_t|\, Z_t) \\
&= \Big[\sum_{Z_{t-1}} \alpha_{t-1} P(Z_t|\, Z_{t-1})\Big] P(X_t|\, Z_t)
\end{aligned}$$

where the sum is over the hidden states at time $t-1$.
Degirmenci [2014] followed a different approach: The **prediction** step calculates the one-step ahead predictive density

$$P(Z_t = j|\, X_{1:t-1}) = \sum_{i=1}^{K} P(Z_t = j|\, Z_{t-1} = i) P(Z_{t-1} = i|\, X_{1:t-1})$$

which acts as the new prior at time t. In the update state, the observed data from time t is absorbed by using Bayes rule:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}
\alpha_t(j) &= P(Z_t = j|\, X_{1:t}) \\
&= P(Z_t = j|\, X_t, X_{1:t-1}) \\
&= \frac{P(X_t = j|\, Z_t = j, X_{1:t-1})P(Z_t = j|\, X_{1:t-1})}{\sum_j P(X_t|\, Z_t = j, X_{1:t-1})P(Z_t = j|\, X_{1:t-1})} \\
&= \frac{1}{C_t} P(X_t|\, Z_t = j)P(Z_t = j|\, X_{1:t-1})
\end{aligned}$$

Note, the observations $X_{1:t-1}$ cancel out since they are d-separated from X_t. Also, C_t is the normalisation constant

$$C_t = P(X_t|\, X_{1:t-1}) = \sum_{j=1}^{K} P(X_t|\, Z_t = j)P(Z_t = j|\, X_{1:t-1})$$

Here, $\alpha_t(j)$ is the forward probability variable for the state j at time t given the observation sequence $X_{1:t}$. The K-vector $\alpha_t = P(Z_t|\, X_{1:T})$ is called the **filtered belief state** at time t. Using matrix notation, the recursive update can be written as

$$\alpha_t \propto \phi_t \circ \left(A^\top \alpha_{t-1}\right)$$

where $\phi_t = [\phi_{t,1}, ..., \phi_{t,K}] = \{P(X_t|\, Z_t = i); 1 \leqslant i \leqslant K\}$ is the local evidence at time t, A is the transition matrix, and $\circ$ is the Hadamard product [4]. The logarithm probability of the evidence is computed as follows:

$$\log P(X_{1:N}|\, \lambda) = \sum_{t=1}^{N} \log P(X_t|\, X_{1:t-1}) = \sum_{t=1}^{N} \log C_t$$

This is the logarithm of the probability of the observation sequence given the HMM model λ. $P(X_{1:N}|\, \lambda)$ is the probability of the observations given the model.

12.2.2.2 The forward-backward algorithm

In the previous section, we have computed the filtered belief states α from the forward messages. We can now compute the backward messages to get the smoothed marginals (see Degirmenci [2014])

$$\begin{aligned}
\gamma_t(j) \equiv P(Z_t = j|\, X_{1:N}) &\propto P(Z_t = j, X_{t+1:N}|X_{1:t}) \\
&\propto P(Z_t = j|\, X_{1:t})P(X_{t+1:N}|\, Z_t = j, X_{1:t})
\end{aligned}$$

[4] Given two matrices A and B, their multiplication AB is defined as

$$AB[i,j] = \sum_k A[i,k]B[k,j]$$

where the multiplication AB makes sense if and only if the number of columns of A are the same as the number of rows of B. This definition agrees with the notion of composition of linear operators.
The Hadamard product is the entrywise multiplication of two matrices, which makes sense when both matrices have the same number of rows and columns. It is defined as

$$(A \circ B)[i,j] = A[i,j]B[i,j]$$

and the related operation adds up the entries of this Hadamard product

$$(A \bullet B)[i,j] = \sum_{i,j} A[i,j]B[i,j]$$

where $A \bullet B$ is a scalar and not a matrix.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

which is the probability of being in state $Z_{t,j}$ given the observations $X_{1:N}$. Since the hidden state at time t is j, the conditional likelihood of future evidence (or observation) is defined as

$$\beta_t(j) \equiv P(X_{t+1:N}|\, Z_t = j)$$

The desired smoothed posterior marginal is defined as

$$\gamma_t(j) \equiv P(Z_t = j|\, X_{1:N})$$

Hence, the smoothed marginals become

$$\gamma_t(j) \propto \alpha_t(j)\beta_t(j)$$

Following Rabiner et al. [1986], the conditional probability of the observations $X_{t+1:T}$ given Z_t is

$$\begin{aligned}
\beta_t &= P(X_{t+1:T}|\, Z_t) \\
&= \sum_{Z_{t+1}} P(X_{t+2:T}|\, Z_{t+1})P(Z_{t+1}|\, Z_t)P(X_{t+1}|\, Z_{t+1}) \\
&= \sum_{Z_{t+1}} \beta_{t+1}P(Z_{t+1}|\, Z_t)P(X_{t+1}|\, Z_{t+1})
\end{aligned}$$

Note, the expected value of $f(Z)$ with respect to the posterior distribution of Z is given by

$$< f(Z) >= E_{Z\sim P(\cdot|X,\lambda)}[f(Z)] = \int_Z f(Z)P(Z|\, X, \lambda)dZ$$

Hence, we can write

$$< Z_t >= \sum_{j=1}^{N} P(Z_t = j|\, X_{1:N}, \lambda)$$

As a result, we get

$$< Z_{t,i} >= \gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{T} \alpha_t(j)\beta_t(j)}$$

which is normalised to get a stochastic matrix.
Following Degirmenci [2014] we have: the terms β are computed recursively from right to left as

$$\begin{aligned}
\beta_{t-1}(i) &= P(X_{t:N}|\, Z_{t-1} = i) \\
&= \sum_{j=1}^{N} P(Z_t = j, X_t, X_{t+1:N}|\, Z_{t-1} = i) \\
&= \sum_{j=1}^{N} P(X_{t+1:N}|\, Z_t = j, X_t, Z_{t-1} = j) \cdots P(Z_t = j, X_t|\, Z_{t-1} = i) \\
&= \sum_{j=1}^{N} P(X_{t+1:N}|\, Z_t = j)P(X_t|\, Z_t = j, Z_{t-1} = i) \cdots P(Z_t = j|\, Z_{t-1} = i) \\
&= \sum_{j=1}^{N} \beta_t(j)\phi_t(j)A_{ij}
\end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

It can be written compactly as

$$\beta_{t-1} = A\big(\phi_t \circ \beta_t\big)$$

The base for β_N is

$$\beta_N(i) = P(X_{N+1:N}|\, Z_N = i) = P(\varnothing|\, Z_N = i) = 1$$

Eventually, we get the smoothed posterior as

$$\gamma_i = \frac{\alpha_i \circ \beta_i}{\sum_j \big(\alpha_i(j) \circ \beta_i(j)\big)}$$

Note, the denominator ensures that each column of γ sums to one, to get a stochastic matrix.

12.2.3 The Viterbi algorithm

We want to compute the most likely sequence of hidden states $Z_1, ..., Z_N$, given the observations $X_1, ..., X_N$ and the model λ. The Viterbi algorithm (see Viterbi [1967]) has a

- forward pass: use the max-product algorithm.
- backward pass: recovers the most likely path through the trellis diagram using a traceback procedure, propagating the most likely state at time t back in time to recursively find the most likely sequence between time $1:t$.

The backward pass can be written as

$$\delta_t(j) \equiv \max_{Z_1,...,Z_{t-1}} P(Z_{1:t-1}, Z_t = j|\, X_{1:t})$$

This probability can be expressed as a combination of the transition from the previous state i at time $t-1$ and the most likely path leading to i,

$$\delta_t(j) = \max_{1 \leqslant i \leqslant K} \delta_{t-1}(i) A_{ij} B_{X_t}(j)$$

where $B_{X_t}(j)$ is the emission probability of observation X_t given state j. Note, we need to keep track of the most likely previous state i,

$$a_t(j) = \arg\max_i \delta_{t-1}(i) A_{ij} B_{X_t}(j)$$

The initial probability is

$$\delta_1(j) = \pi_j B_{X_1(j)}$$

The most likely final state is

$$Z_N^* = \arg\max_i \delta_N(i)$$

The most likely sequence can be computed using traceback as

$$Z_t^* = a_{t+1} Z_{t+1}^*$$

In the logarithm space, we have $\log\max = \max\log$, so that we express the probability as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\log \delta_t(j) \equiv \max_i \log \delta_{t-1}(i) + \log A_{ij} + \log B_t(j)$$

Note, in problems with large parameter spaces, the transition matrix ends up being very large. This requires a large data set to avoid severe overfitting. One solution is to use factorial or hierarchical HMMs.

12.2.4 Estimating parameters: The Baum-Welch algorithm

We consider the problem of estimating the state transition matrix A and the emission probability B, given a single sequence of observations $X_{1:N}$. We use the Baum-Welch algorithm (see Baum [1972]), which is an instance of the Expectation-Maximisation (EM) (see Dempster et al. [1977]) described in Section (12.2.1.2). One starts by randomly initialising the values for A and B, and then repeatedly updating them. Each updating iteration has two steps: (1) the Expectation step (E-step), (2) the Maximisation step (M-step).
Given a sequence of observations $X_{1:N}$, we want to compute

$$\arg\max_\lambda P(X;\lambda) = \arg\max_Z P(X,Z;\lambda)$$

by performing the maximum-likelihood estimation. The summation being computationally intensive, we use the Expectation-Maximisation (EM) to estimate the model parameters. We must get the forward and backward probabilities α, β calculated by using the forward-backward algorithm. Murphy proposed a two-step process:

1. E-step:

$$\gamma_{t,k} \equiv P(Z_{t,k} = 1 |\, X, \lambda_{old}) = \frac{\alpha_k(t)\beta_k(t)}{\sum_{j=1}^N \alpha_j(t)\beta_j(t)}$$

   and

$$\xi_{t,jk} \equiv P(Z_{t-1,j} = 1, Z_{t,k} = 1 |\, X, \lambda_{old}) = \frac{\alpha_j(t)A_{jk}\beta_k(t+1)B_k(X_{t+1})}{\sum_{i=1}^N \alpha_i(t)\beta_i(t)}$$

2. M-step: The parameter estimation problem is turned into a constraint optimisation problem where $P(X_{1:N} |\, \lambda)$ is maximised, subject to the stochastic constraints of the HMM parameters. The Lagrange multiplier (see Section (18.5)) can be used to find the model parameters. We get the following results:

$$\hat{\pi}_k = \frac{E[N_k^1]}{N} = \frac{\gamma_{1,k}}{\sum_{j=1}^K \gamma_{1,j}}$$

$$\hat{A}_{jk} = \frac{E[N_{j,k}]}{\sum_{k'} E[N_{j,k}]} = \frac{\sum_{t=2}^N \xi_{t,jk}}{\sum_{l=1}^K \sum_{t=2}^N \xi_{t,jl}}$$

$$\hat{B}_{jl} = \frac{E[M_{jl}]}{E[N_j]} = \frac{\sum_{t=1}^N \gamma_{t,l} X_{t,j}}{\sum_{t=1}^N \gamma_{t,l}}$$

   Then, the new model parameters are

$$\lambda_{new} = (\hat{A}, \hat{B}, \hat{\lambda})$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

12.3 Some generative models

12.3.1 Overview

Autoencoders (AEs) are neural networks used to learn efficient codings of unlabelled data in unsupervised learning (see Kramer [1991], Hinton et al. [1994]). The encoding is validated and refined by attempting to regenerate the input from the encoding. An autoencoder has two main parts: an encoder that maps the input into the code, and a decoder that maps the code to a reconstruction of the input. The simplest form of an autoencoder is a feedforward, non-recurrent neural network that participate in multilayer perceptrons (MLP) by employing an input layer and an output layer connected by one or more hidden layers. The output layer has the same number of nodes as the input layer and its purpose is to reconstruct its inputs. This is illustrated in Figure (12.3). The autoencoder learns a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore insignificant data called noise. Variants exist, aiming to force the learned representations to assume useful properties. Traditionally applied to dimensionality reduction or feature learning, the concept became widely used for learning generative models of data (see Kingma et al. [2014]). Modern variations of AEs have been applied to other tasks such as anomaly detection, image processing, drug discovery, machine translation, among others. We present their mathematical formulation in Section (12.4).

Figure 12.3: Autoencoder.

As discussed in Section (11.2.4.2), a generative model is a statistical model of the joint probability distribution of two observed random variables. In general, generative models focus on providing a parametric specification of a probability distribution function (pdf). These models can then be trained by maximising the log-likelihood (see Section (11.2.2.2)). However, in these models, the pdf is generally intractable, requiring numerical approximation to compute the gradient. This is the case of the deep Boltzmann machine (see Salakhutdinov et al. [2009]). Recently, deep generative models (DGMs) were developed through the combination of generative models and deep neural networks by approximating the pdf via deep neural networks from learned samples. They represent the log-likelihood implicitly while generating samples from the desired distribution. For instance, generative stochastic networks can be trained with exact backpropagation, but rely on Markov chains for computation. Kingma et al. [2014] and Rezende et al.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2014] developed more general stochastic backpropagation rules, allowing one to backpropagate through Gaussian distributions with finite variance, and to backpropagate to the covariance parameter as well as the mean. They trained variational autoencoders (VAEs), which combines a differentiable generator network with a second neural network. The main idea being that given one random variable z with one distribution, we can create another random variable $x = g(z)$ with very different distribution. The deterministic function g is learned from the data through the function approximation capacity of neural networks. VAEs rely on the maximum likelihood by adjusting the generative process via backpropagation to maximise the (unknown) probability of observing the given training samples:

$$p_\theta(x) = \int p_\theta(x|\, z) p_\theta(z) dz$$

where z is a latent variable [5] and θ is a vector of network parameters. We need to define the latent variable space $\mathcal{Z}$ to capture the relevant information in the generative process, and take the integral over z. In the former, the authors relied on the inherent property of neural networks as functional approximators. They chose the initial distribution as a d-dimensional Gaussian:

$$\begin{aligned} p_\theta(x|\, z) &= N(z; \mu, \sigma^2 I) \\ p_\theta(z) &= N(z; 0, I) \end{aligned}$$

where I is the identity matrix, μ is the mean, and σ is the standard deviation. We can then find a continuously differentiable expression for $p_\theta(x)$. We optimise the model by using stochastic gradient descent to update the network parameters θ. Thus, VAEs require differentiation through the hidden units.

Schmidhuber [1992b] proposed predictability minimisation having two neural networks compete. Each hidden unit in a neural network is trained to be different from the output (a single scalar) of a second network, which predicts the value of that hidden unit given the value of all of the other hidden units. Thus, it is a regulariser encouraging the hidden units of a neural network to be statistically independent, while they accomplish some other task. It is an optimisation problem with an objective function to be minimised, and learning approaches the minimum of the objective function. Following a similar approach, Goodfellow et al. [2014] proposed GAN models which eliminate Markov chains by observing that

$$\lim_{\sigma \to 0} \nabla_x E_{\epsilon \sim N(0, \sigma^2 I)}[f(x + \epsilon)] = \nabla_x f(x)$$

where σ is the standard deviation treated as a hyperparameter. GANs require differentiation through the visible units, and thus cannot model discrete data, while VAEs require differentiation through the hidden units, and thus cannot have discrete latent variables. In GANs, one network produces a rich, high dimensional vector that is used as the input to another network, and attempts to choose an input that the other network does not know how to process. Here, the competition between the networks is the only training criterion. It is based on a minimax game with a value function that one agent seeks to maximise and the other seeks to minimise. The game terminates at a saddle point corresponding to a minimum for one player and a maximum for the other player. GANs are implicit generative models, which means that they do not explicitly model the likelihood function nor provide means for finding the latent variable corresponding to a given sample (see Mohamed et al. [2016]).

12.3.2 Autoencoding variational Bayes

Salimans et al. [2013] used re-parametrisation in an efficient version of a stochastic variational inference algorithm for learning the natural parameters of exponential-family approximating distributions. Kingma et al. [2014] showed that for i.i.d. datasets with continuous latent variables per datapoint, posterior inference could be made efficient

[5] Latent variables are variables that are not directly observed but are rather inferred (through a mathematical model) from other variables that are observed (directly measured).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

by fitting an approximate inference model to the intractable posterior by using the lower bound estimator. They proposed the Auto-Encoding Variational Bayesian (AEVB) algorithm, which uses the SGVB estimator to optimise a recognition model $q_\phi(z|x)$. This way, they performed very efficient approximate posterior inference using simple ancestral sampling, allowing them to learn the model parameters without the need of expensive iterative inference schemes (such as MCMC) per datapoint. When a neural network is used for the recognition model, they obtained the variational auto-encoder (VAE).

12.3.2.1 Description

12.3.2.1.1 The problem We consider a dataset $X = \{x^1, ..., x^N\}$ of N i.i.d. samples of some continuous or discrete variable x. Assume that the data is generated in two steps by some random process involving an unobserved continuous random variable z as follows:

1. A value z^i is generated from some prior distribution $p_{\theta*}(z)$
2. A value x^i is generated from some conditional distribution $p_{\theta*}(x|z)$

Both distributions $p_{\theta*}(z)$ and $p_{\theta*}(x|z)$ come from parametric families of distributions $p_\theta(z)$ and $p_\theta(x|z)$, respectively, which are differentiable almost everywhere w.r.t. both θ and z. Note, from Equation (11.2.5), $p_\theta(x, z) = p_\theta(z)p_\theta(x|z)$ is the generative model with parameters θ. Further, the true parameters θ^* and the latent variables z^i are unknown. The authors considered an algorithm that works efficiently in the case of:

1. Intractability: when the integral of the marginal likelihood

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$$

   is intractable, the true posterior density $p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}$ is intractable, and where the required integrals for any reasonable mean-field variational Bayesian (VB) algorithm are also intractable.

2. A large dataset: sampling-based solutions such as Monte Carlo EM are too slow.

The authors proposed to consider three types of problems:

1. an approximate maximum likelihood estimation of the parameter θ,
2. an approximate posterior inference of the latent variable z given an observed value x for a choice of parameters θ,
3. an approximate marginal inference of the variable x.

To do so, they introduced the recognition model (or approximate posterior) $q_\phi(z|x)$ (see Equation (11.2.3)), where ϕ are the variational parameters, which is an approximation to the intractable true posterior density $p_\theta(z|x)$. They introduced a method for learning the recognition model parameters ϕ jointly with the generative model parameters θ. Since the unobserved variables z are latent variables, the distribution $q_\phi(z|x)$ is a probabilistic encoder. Similarly, the conditional distribution $p_\theta(x|z)$ is a probabilistic decoder, since given a code z it produces a distribution over the possible corresponding values of x.

The marginal likelihood (see Section (11.2.3.2)) is composed of a sum over the marginal likelihoods of individual datapoints

$$\log p_\theta\left(x^1, ..., x^N\right) = \sum_{i=1}^{N} \log p_\theta(x^i)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

which can each be rewritten as (see Equation (11.2.4))

$$\log p_\theta(x^i) = D_{KL}\big(q_\phi(z|x^i)||p_\theta(z|x^i)\big) + \mathcal{L}(\theta, \phi; x^i)$$

where D_{KL} is the Kullback-Leibler distance (see Definition (2.2.4)), that is, the KL divergence of the approximate from the true posterior, and $\mathcal{L}(\theta, \phi; \cdot)$ is the (variational) lower bound on the marginal likelihood of datapoint i. It can be written as

$$\mathcal{L}(\theta, \phi; x^i) = -D_{KL}\big(q_\phi(z|x^i)||p_\theta(z)\big) + E_{q_\phi(z|x^i)}[\log p_\theta(x^i|z)]$$

which we want to differentiate and optimise w.r.t. both the variational parameters ϕ and the generative parameters θ. However, computing the gradient of the lower bound w.r.t. ϕ is difficult.

12.3.2.1.2 The solution The authors showed that a reparameterisation of the variational lower bound yielded a simple differentiable unbiased estimator of the lower bound. The strategy to derive lower bound estimator for directed graphical models is illustrated in Figure (12.4).

Figure 12.4: Strategy to derive lower bound estimator for directed graphical models. Solid lines denote the generative model $p_\theta(z)p_\theta(x|z)$, dashed lines denote the variational approximation $q_\phi(z|x)$.

12.3.2.2 The SGVB

Under certain conditions for the approximated posterior $q_\phi(z|x)$, the authors re-parametrised the random variable $\tilde{z} \sim q_\phi(z|x)$ by using a differentiable transformation $g_\phi(x, \epsilon)$ of an (auxiliary) noise variable ϵ:

$$\tilde{z} = g_\phi(x, \epsilon) \text{ with } \epsilon \sim p(\epsilon)$$

Then, they formed Monte Carlo estimates of expectations of some function $f(z)$ w.r.t. $q_\phi(z|x)$. Applying this technique to the variational lower bound, they got the SGVB (Stochastic Gradient Variational Bayes) estimators $\tilde{\mathcal{L}}^A(\theta, \phi; \cdot) \approx \mathcal{L}(\theta, \phi; \cdot)$ and $\tilde{\mathcal{L}}^B(\theta, \phi; \cdot) \approx \mathcal{L}(\theta, \phi; \cdot)$ which can be used for efficient approximate posterior inference in almost any model with continuous latent variables and/or parameters, and can be optimised by using standard stochastic gradient ascent techniques. They are given by

$$\tilde{\mathcal{L}}^A(\theta, \phi; x^i) = \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(x^i, z^{i,l}) - \log q_\phi(z^{i,l}|x^i)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

and

$$\tilde{\mathcal{L}}^B(\theta,\phi;x^i) = -D_{KL}\big(q_\phi(z|x^i)||p_\theta(z)\big) + \frac{1}{L}\sum_{l=1}^{L}\big(\log p_\theta(x^i|z^{i,l})\big) \tag{12.3.13}$$

where $z^{i,l} = g_\phi(\epsilon^{i,l}, x^i)$ and $\epsilon^l \sim p(\epsilon)$. This equation corresponds to an auto-encoder where the first term acts as a regulariser, and the second term is a an expected negative reconstruction error. This is illustrated in Figure (12.5).

Figure 12.5: VAE graph.

Given multiple datapoints from a dataset X with N datapoints, they constructed an estimator of the marginal likelihood lower bound of the full dataset, based on minibatches

$$\mathcal{L}(\theta,\phi;X) \approx \tilde{\mathcal{L}}^M(\theta,\phi;X^M) = \frac{N}{M}\sum_{i=1}^{M}\tilde{\mathcal{L}}(\theta,\phi;x^i)$$

where the minibatch $X^M = \{x^1,...,x^M\}$ is a randomly drawn sample of M datapoints from the full dataset X with N datapoints. The derivatives $\nabla_{\theta,\phi}\tilde{\mathcal{L}}^M(\theta,\phi;X^M)$ can be taken, and the resulting gradients can be used in conjunction with stochastic optimisation methods. The Auto-Encoding VB (AEVB) algorithm is basic approach to compute the stochastic gradients.

12.3.2.3 The reparametrisation

The authors used an alternative method for generating samples from the recognition model $q_\phi(z|x)$. Let z be a continuous random variable, and $z \sim q_\phi(z|x)$ be some conditional distribution. They expressed the random variable z as a deterministic variable $z = g_\phi(x,\epsilon)$, where ϵ is an auxiliary variable with independent marginal distribution $p(\epsilon)$, and g_ϕ is some vector-valued function parameterised by ϕ. It is used to rewrite an expectation w.r.t $q_\phi(z|x)$ such that the Monte Carlo estimate of the expectation is differentiable w.r.t. ϕ. The differentiable estimator is constructed as follows:

$$E_{q_\phi(z|x)}[f(z)] = \int q_\phi(z|x)f(z)dz \approx \frac{1}{L}\sum_{i=1}^{L} f\big(g_\phi(x,\epsilon^l)\big)$$

where $\epsilon^l \sim p(\epsilon)$.
As an example, in the univariate Gaussian case we get $z \sim p(z|x) = N(\mu,\sigma^2)$. Then, a valid reparameterisation is given by $z = \mu + \sigma\epsilon$, where ϵ is a white noise $\epsilon \sim N(0,1)$. In that setting, the estimator becomes

$$E_{N(z;\mu,\sigma^2)}[f(z)] = E_{N(\epsilon;0,1)}[f(\mu+\sigma\epsilon)] \approx \frac{1}{L}\sum_{i=1}^{L} f(\mu+\sigma\epsilon^l)$$

$\epsilon^l \sim N(0,1)$. Note, there are different methods for computing $q_\phi(z|x)$: (1) Tractable inverse CDF, (2) Composition, among others.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

12.3.2.4 The variational AE

Here, the authors used a neural network for the probabilistic encoder $q_\phi(z|x)$, where the parameters ϕ and θ are optimised jointly with the AEVB algorithm. They let the prior over the latent variables be the centered isotropic multivariate Gaussian $p_\theta(z) = N(z; 0, I)$ and the conditional distribution $p_\theta(x|z)$ be a multivariate Gaussian (in case of real-valued data) or Bernoulli (in case of binary data) whose distribution parameters are computed from z with a MLP (a fully-connected neural network with a single hidden layer) (see Section (5.1.3)). In this setting, the true posterior $p_\theta(z|x)$ is intractable. Assume it takes on a approximate Gaussian form with an approximately diagonal covariance. Thus, let the variational approximate posterior be a multivariate Gaussian with a diagonal covariance structure:

$$\log q_\phi(z|x^i) = \log N\big(z; \mu_i, \sigma_i^2 I\big)$$

where the mean and standard deviation of the approximate posterior, μ_i and σ_i, respectively, are outputs of the encoding MLP. We sample from the posterior $z^{i,l} \sim q_\phi(z|x^i)$ by using $z^{i,l} = g_\phi(x^i, \epsilon^l) = \mu_i + \sigma_i \odot \epsilon^l$, where $\epsilon^l \sim N(0, I)$ and the operator $\odot$ denotes the Hadamard product [6]. Here, both the prior $p_\theta(z)$ and the approximate posterior $q_\phi(z|x)$ are Gaussian, so that we can use the estimator in Equation (12.3.13), where the KL divergence can be computed and differentiated without estimation. The resulting estimator for this model and datapoint is

$$\mathcal{L}(\theta, \phi; X) \approx \frac{1}{2}\sum_{i=1}^{J}\big(1 + \log \sigma_{i,j}^2 - \mu_{i,j}^2 - \sigma_{i,j}^2\big) + \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(x^i|z^{i,l})$$

where $z^{i,l} = \mu_i + \sigma_i \odot \epsilon^l$ and $\epsilon^l \sim N(0, I)$, and the decoding term $p_\theta(x^i|z^{i,l})$ is a Bernoulli or Gaussian MLP, depending on the type of data we are modelling. This is illustrated in Figure (12.6).

Figure 12.6: VAE graph.

12.3.3 Generative adversarial networks

12.3.3.1 Description

A generative adversarial network (GAN) model (see Goodfellow et al. [2014]) is a novel approach for training generative models. They estimate the latter via an adversarial process where two models are simultaneously trained:

[6] Given two matrices A and B, the Hadamard product is the entry-wise multiplication of two matrices, which makes sense when both matrices have the same number of rows and columns. It is defined as

$$(A \circ B)[i, j] = A[i, j]B[i, j]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1. Generator: A generative net G captures the data distribution by learning to confuse the discriminator D by generating high quality data. It is acting as a team of counterfeiters.

2. Discriminator: A discriminative net D that estimates the probability that a sample came from the training data (real) rather than G. It learns to separate the data created by G from the observed data. It acts as the police trying to catch the counterfeiters.

Given a training set, this technique learns to generate new data with the same statistics as the training set. The generative network, G, generates candidates while the discriminative network, D, evaluates them. Typically, the generative network learns to map from a latent space [7] to a data distribution of interest, while the discriminative network distinguishes candidates produced by the generator from the true data distribution. The training process for G is to maximise the probability that D makes a mistake (incapacity of recognising true from generated data), which corresponds to a minmax two-player game. In order for G to confuse D as much as possible, it must reduce the adversarial loss in the sense that D will not correctly discriminate the prediction. The target loss for D is equal to the adversarial loss on D (D is able to classify correctly inputs from G as fake and the dataset as true). Through competition, both networks improve their performance until the genuine data is indistinguishable from the generated one. If the pair (G, D) is made of multilayer perceptrons, the system can be trained with backpropagation (see Section (5.3)). This is illustrated in Figure (12.7).

Figure 12.7: GAN graph.

We present the adversarial nets, where both G and D are multilayer perceptrons, and G generates samples by passing random noise as input. We let z be the input noise variables with prior distribution $p_Z(z)$, and want to learn (implicitly) the probability distribution p_G over data x. Here, we have:

- The generator is the neural network $G(z, \theta_G)$ with the parameter θ_G. It maps data to the data-input space $G : z \to x$.

- The Discriminator is the neural network $D(x, \theta_D)$ that scores the likelihood that x was sampled from the known dataset, $p_{data}(x) - D : x \to [0, 1]$.

The Discriminator D is trained to maximise correct labelling via the uncertainty measure $\log D(x)$ (see Equation (2.2.1)), while G is trained to minimise $\log(1 - D(G(z)))$. Note, there is no explicit representation for the distribution $p_G(x)$ since it is expressed via $G(\cdot, \theta_G)$. As such, Markov chains are not needed.
Thus, the pair (G, D) plays a two-player minmax game with value function $V(G, D)$ given by

$$\min_G \max_D V(G, D) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_Z(z)}[\log(1 - D(G(z)))] \tag{12.3.14}$$

[7] A latent space, also known as a latent feature space or embedding space, is an embedding of a set of items within a manifold in which items which resemble each other more closely are positioned closer to one another in the latent space. Position within the latent space can be viewed as being defined by a set of latent variables that emerge from the resemblances from the objects.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The first term is the entropy (see Equation (2.2.2)) in information theory (IT). To better understand the characteristics of the value function, we display the plot of the logarithm function in Figure (12.8). We see that maximising D and minimising G leads to contradictory result for the value function.

Figure 12.8: Plot of the logarithm function.

The game is implemented by using an iterative, numerical approach. This is detailed in Algorithm (29). First, the input noise z is uniformly sampled, and then the mapping $x = G(z)$ imposes the non-uniform distribution p_G on the transformed samples. G contracts in regions of high density and expands in regions of low density. The gradient of D guides $G(z)$ to flow to regions that are more likely to be classified as data. Eventually, if the pair (G, D) has enough capacity, it will reach a saddle point where $D(x) = \frac{1}{2}$, meaning that $p_G = p_{data}$.
Since optimising D in an inner loop is expensive, the authors proposed to alternate between k steps of optimising D and one step of optimising G, to maintain D near its optimal solution as long as G slowly evolves. In the early stage of the process, G is poor and D has a high accuracy at rejecting the samples created from G, leading $\log(1 - D(G(z)))$ to saturate. One solution is to train G to maximise $\log D(G(z))$, which provides stronger gradient in the early stage.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 29 Minibatch stocastic gradient descent training of GAN.

Require: The number of steps to apply to the discriminator k is a hyperparameter. The least expensive option is $k = 1$.

Require: get the number of training iteration N.

for N **do**
  for k steps **do**
    Sample minibatch of m noise samples $\{z^1, ..., z^m\}$ from noise prior $p_G(z)$.
    Sample minibatch of m examples $\{x^1, ..., x^m\}$ from data generating distribution $p_{data}(x)$.
    Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_D} = \frac{1}{m}\sum_{i=1}^{m}\left[\log D(x^{(i)}) + \log\left(1 - D(G(z^{(i)}))\right)\right]$$

  end for
  Sample minibatch of m noise samples $\{z^1, ..., z^m\}$ from noise prior $p_G(z)$.
  Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_G} = \frac{1}{m}\sum_{i=1}^{m}\log\left(1 - D(G(z^{(i)}))\right)$$

end for

12.3.3.2 Analysis

From Algorithm (29) we see that the generator G implicitly defines a probability distribution p_G obtained by the mapping $G : z \to x$ for $z \sim p_Z$. We now discuss convergence to saddle point where $D(x) = \frac{1}{2}$. See the logarithm function in Figure (12.8). The authors first showed that the minmax problem has a global optimum for $p_G = p_{data}$.

***Proposition* 1** *For G fixed, the optimal discriminator D is*

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

Proof: Given a generator G, from Equation (12.3.14), D maximise the quantity

$$\begin{aligned} V(G, D) &= \int_x p_{data}(x)\log(D(x))dx + \int_z p_z(z)\log(1 - D(g(z)))dz \\ &= \int_x p_{data}(x)\log(D(x)) + p_G(x)\log(1 - D(x))dx \end{aligned}$$

For any $(a, b) \in \mathbb{R}^2\backslash\{0, 0\}$, the function $y \to a\log(y) + b\log(1 - y)$ reaches its maximum in $[0, 1]$ at $\frac{a}{a+b}$. The discriminator does not need to be defined outside of $\text{Supp}(p_{data}) \cup \text{Supp}(p_G)$.
This is equivalent to maximising the log-likelihood for estimating the conditional probability $P(Y = y|x)$, where Y indicates whether x comes from p_{data} (with $y = 1$) or from p_G (with $y = 0$). Thus, the minmax game in Equation (12.3.14) can be rewritten as

$$C(G) = \max_D V(G, D) = E_{x\sim p_{data}}\left[\frac{p_{data}(x)}{p_{data}(x) + p_G(x)}\right] + E_{x\sim p_G}\left[\frac{p_G(x)}{p_{data}(x) + p_G(x)}\right]$$

Theorem 12.3.1 *The global minimum of the virtual training criterion $C(G)$ is reached if and only if $p_G = p_{data}$. At that point, $C(G)$ reaches the value $-\log 4$.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Proof: For $p_G = p_{data}$, $D_G^* = \frac{1}{2}$. Hence, from the above equation at $D_G^* = \frac{1}{2}$, we get $C(G) = \log \frac{1}{2} + \log \frac{1}{2} = -\log 4$. This is the best possible value of $C(G)$, reached only for $p_G = p_{data}$, since

$$E_{x \sim p_{data}}[-\log 2] + E_{x \sim p_G}[-\log 2] = -\log 4$$

and by subtracting this expression from $C(G) = V(G, D_G^*)$, we get

$$C(G) = -\log 4 + KL(p_{data} \| \frac{p_{data} + p_G}{2}) + KL(p_G \| \frac{p_{data} + p_G}{2})$$

where $KL(\cdot)$ is the Kullback-Leibler divergence. The above expression is the Jensen-Shannon divergence JSD between the model's distribution and the data generating process:

$$C(G) = -\log 4 + 2 \cdot JSD(p_{data} \| p_G)$$

Since the JSD between two distributions is always non-negative, and zero if and only if they are equal, the authors showed that $C^* = -\log 4$ is the global minimum of $C(G)$ and that the only solution if $p_G = p_{data}$.
The convergence of Algorithm (29) is given as follows:

***Proposition* 2** *If the pair (G, D) have enough capacity, and at each step of Algorithm (29), the discriminator is allowed to reach its optimum given G, and p_G is updated so as to improve the criterion*

$$E_{x \sim p_{data}}[\log D_G^*(x)] + E_{x \sim p_{data}}[\log (1 - D_G^*(x))]$$

then p_G converges to p_{data}.

Proof: see the authors.

***Remark* 12.3.1** *Note, in practice, adversarial nets are limited family of p_G distributions via the function $G(z, \theta_G)$, and we optimise θ_G instead of p_G directly. Hence, the above proof does not apply. The authors rely on the good performance of multiplier perceptrons to suggest that they are reasonable models to use despite their lack of theoretical guarantees.*

12.3.3.3 Improvements

In general, GANs are good at learning an unconditional model such as image and text creation or discovery. However, when it comes to learning financial time series, one require a sampling process taking into account the previous state space to preserve the statistical properties of the series. As a solution, Goodfellow et al. [2014] proposed to consider Conditional GANs which can handle dependent data generation. In that setting, the conditional distribution $p(x|c)$ can be obtained by adding c as input to the pair (G, D). Further, they showed that one could perform learned approximate inference by training an auxiliary network to predict z given x. While similar to the inference net trained by the wake-sleep algorithm (see Hinton et al. [1995]), here the inference net may be trained for a fixed generator net after the generator net has finished training.
Over time new architectures, loss functions, training, among others, developed to improve the original model. Some of these improvements are as follows:

- Ensemble strategies: Train multiple GANs with different initial conditions, slices of data and tasks to attain. The output from the generators are aggregated with some operators (summing, weighted average) from multiple checkpoints or at the end of the training. Examples of these steps are Stacked GANs (see Huang et al. [2017]), Ensembles of GANs (see Wang et al. [2016]), AdaGANS (see Tolstikhin et al. [2017]), among others.
- Loss function reshaping: Reshape the original loss function to circumvent problems related to training instability.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Adjusting architecture and training process: As an example, the Deep Convolutional GAN (see Radford et al. [2015]) has a set of constraints on the architectural topology of Convolutional GANs to make the training more stable. Another example is the Evolutionary GANs (see Wang et al. [2018]) that adds to the training loop of a GAN different metrics to jointly optimise the generator. It also employs a population of generators, created by introducing novel mutation operators.
- Bidirectional GAN (BiGAN): It aims at introducing a generator model to act as the discriminator, whereby the discriminator naturally considers the entire translation space so that the inadequate training problem can be alleviated (see Zhang et al. [2018]). To satisfy this property, generator and discriminator are both designed to model the joint probability of sentence pairs, with the difference that, the generator decomposes the joint probability with a source language model and a source-to-target translation model, while the discriminator is formulated as a target language model and a target-to-source translation model. To further leverage the symmetry of them, an auxiliary GAN is introduced and adopts generator and discriminator models of original one as its own discriminator and generator respectively. Two GANs are alternately trained to update the parameters.

12.3.3.4 Conditional generative adversarial networks

Conditional GANs (cGANs) are an extension of GANs (see Mirza et al. [2014]), where the pair (G, D) is not only based in noise or generated inputs, but includes an additional information set V. It can represent a class label, a certain categorical feature, or an expected market condition. That is, cGANs try to learn an implicit conditional generative model. It can be constructed by feeding the data we wish to condition on, to both the generator and the discriminator. We let v be the conditional variable and modify the GAN model as follows: $G : z \times v \rightarrow x$ and $D : x \times v \rightarrow [0, 1]$. Now, G is trained to minimise $\log\big(1 - D(G(z|v))\big)$. Again, the pair (G, D) plays a two-player minmax game with value function $V(G, D)$ given by

$$\min_G \max_D V(G, D) = E_{x\sim p_{data}(x)}[\log D(x|\, v)] + E_{z\sim p_Z(z)}[\log\big(1 - D(G(z|\, v))\big)]$$

For example, when forecasting a time series $y_1, y_2, ..., y_T$, the conditional set is $V = \{y_{t-1}, y_{t-2}, ..., y_{t-p}\}$, and we want to discriminate $x = y_t$ with $p_{data}(y_t\,|y_{t-1}, y_{t-2}, ..., y_{t-p})$. Here, p sets the amount of past information in the implicit conditional generative model. For $p = 0$ we recover the GAN model. For $p > 0$, the NN needs larger memory and more capacity to select and model the right past values.

12.4 A review of anomaly detection

We consider the data set $\mathcal{X} = \{x_1, ..., x_N\}$ with $x_i \in \mathbb{R}^D$, and we let $\mathcal{Z} \in \mathbb{R}^K$ $(K << N)$ be a representation space. In classification (see Section (3.3.5)), we aim at learning a feature representation mapping function $\phi : \mathcal{X} \rightarrow \mathcal{Z}$, or, a score learning function $\tau : \mathcal{X} \rightarrow \mathbb{R}$. One solution is to let both ϕ and τ be neural network mapping functions with $H \in \mathbb{N}$ hidden layers, and weight matrices $\mathcal{M} = \{M^1, ..., M^H\}$. As an example, in the case of anomaly detection (see review from Pang et al. [2020], Bulusu et al. [2020]), we can either directly use the function τ to infer the anomaly score, or, learn the feature mapping ϕ and perform an additional step to compute a score for each new data instance in the representation space.

12.4.1 Feature extraction

Feature extraction consists in learning to extract low-dimensional feature representations from high-dimensional and (or) non-linearly separable data for some classification applications. That is,

$$z = \phi(x; \mathcal{M})$$

where $\phi : \mathcal{X} \rightarrow \mathcal{Z}$ is a deep network feature mapping function, with $\mathcal{X} \in \mathbb{R}^D$, $\mathcal{Z} \in \mathbb{R}^K$, and $D >> K$. $\mathcal{Z}$ is the latent or feature space. Deep feature extraction models offer more powerful dimensionality reduction than linear models. As

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

an example, one can use pre-trained deep learning models to extract low-dimensional features. There is a large number of pre-trained deep models available in the market. For instance, in image recognition: AlexNet (see Krizhevsky et al. [2017]), VGG (see Simonyan et al. [2015]). These feature representations can then be transferred to be fine-tuned on a classifier. We can also directly train a deep feature extraction model

12.4.2 Learning feature representations of normality

Generic normality feature learning learns the representations of data instances by optimising a generic learning objective function, capturing some key underlying data regularities. It is mathematically written as

$$\{\mathcal{M}^*, W^*\} = \arg\min_{\mathcal{M},W} \sum_{x\in\mathcal{X}} \ell\Big(\psi\big(\phi(x;\mathcal{M});W\big)\Big) \tag{12.4.15}$$

where ϕ maps the original data into the representation space $\mathcal{Z}$, ψ parametrised by W is a surrogate [8] learning task operating on the $\mathcal{Z}$ space dedicated to enforcing the learning of the underlying data regularities, and ℓ is a loss function relative to the underlying model. If the feature space $\mathcal{Z}$ has lower dimensionality than the input space $\mathcal{X}$, the feature vector $\phi(x;\mathcal{M})$ can be regarded as a compressed representation of the input x. This is the case of undercomplete autoencoders. Note, $z = \phi(x;\mathcal{M})$ is usually referred to as code, latent variables, or a latent representation. Further, we can write $\widehat{x} = (\psi\circ\phi)(x) = \psi\big(\phi(x;\mathcal{M});W\big)$, where $\circ$ is the convolution operator, and the loss function becomes $\ell(x,\widehat{x})$.

Various techniques exist to prevent autoencoders from learning the identity function and to improve their ability to capture important information and learn richer representations. This is usually done by adding regularisation function in the loss function. For example, a sparse autoencoder is an autoencoder whose training criterion involves a sparsity penalty $\Omega(\phi)$ on the code layer ϕ:

$$\ell(x,\widehat{x}) + \Omega(\phi)$$

where the penalty encourages the model to activate (output value close to 1) specific areas of the network on the basis of the input data, while inactivating all other neurons (to have an output value close to 0) (see Makhzani et al. [2014]). This sparsity can be achieved by formulating the penalty terms in different ways.

In the case of anomaly detection, the anomaly score s_x is given by

$$s_x = f\big(x, \phi_{\mathcal{M}^*}, \psi_{W^*}\big)$$

where f is scoring function using the above two functions with the trained parameters $\mathcal{M}^*$ and W^* to compute the anomaly score. This approach include methods driven by several perspectives such as data reconstruction, generative modelling, predictability modelling, and self-supervised classification.

12.4.2.1 Autoencoders

Autoencoders (AEs) (see Section (12.3.1)) aim at learning some low-dimensional feature representation space on which the given data instances can be well reconstructed. It can be used for data compression or dimension reduction. The learned feature representations are enforced to learn important regularities of the data to minimise reconstruction errors. As a result, anomalies in the data are difficult to be reconstructed from the obtained representation, exhibiting large reconstruction errors. Hence, these methods are commonly used in anomaly detection.
An AE network is composed of

[8] A surrogate model is an engineering method used when an outcome of interest cannot be easily directly measured, so a model of the outcome is used instead.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1. An encoding network: It maps the original data into low-dimensional feature space. This is the function ϕ in Equation (12.4.15).
2. A decoding network: It attempts at recovering the data from the projected low-dimensional space. This is the function ψ in Equation (12.4.15).

In order to minimise the overall reconstruction error, one requires the retained information to be as much relevant as possible to the dominant instances, called the normal instances. Consequently, one can use the data reconstruction error to fit the anomaly score in anomaly detection. This approach can be formulated as

$$\begin{aligned} z &= \phi_e(x;\mathcal{M}_e) \\ \widehat{x} &= \phi_d(z;\mathcal{M}_d) \\ \{\mathcal{M}_e^*,\mathcal{M}_d^*\} &= \arg\min_{\mathcal{M}_e,\mathcal{M}_d} \sum_{x\in\mathcal{X}} \left\|x-\widehat{x}\right\|^2 \end{aligned}$$

where

- ϕ_e is the encoding network with parameters $\mathcal{M}_e$, and
- ϕ_d is the decoding network with parameters $\mathcal{M}_d$.

Note, writing $\widehat{x} = (\phi_d \circ \phi_e)(x) = \phi_d\big(\phi_e(x;\mathcal{M}_e);\mathcal{M}_d\big)$, we recover the generic learning objective function in Equation (12.4.15) by letting the loss function $\ell(x,\widehat{x}) = \left\|x-\widehat{x}\right\|^2$ be the L_2-norm.
The anomaly score s_x is given by

$$s_x = \|x - \phi_d\big(\phi_e(x;\mathcal{M}_e^*);\mathcal{M}_d^*\big)\|^2$$

Note, the encoder and decoder can share the same weight parameters to reduce the number of parameters and regularise the learning. There are a lot of different types of encoders available in the market. For instance, sparse AE encourages sparsity in the activation units of the hidden layer by (for example) keeping the top K most active units (see Makhzani et al. [2014]). In order to learn feature representations that are robust to small variations, denoising AE was introduced (see Vincent et al. [2010]). The model learns to reconstruct data from some predefined corrupted data instances rather than the original data. The objective of denoising autoencoders is that of cleaning the corrupted input, or denoising. Two assumptions are inherent to this approach: (1) Higher level representations are relatively stable and robust to the corruption of the input, (2) To perform denoising well, the model needs to extract features that capture useful structure in the input distribution. The training process of a DAE works as follows:

- The initial input x s corrupted into $\tilde{x}$ through stochastic mapping $\tilde{x} \sim q_D(\tilde{x}|x)$.
- The corrupted input $\tilde{x}$ is then mapped to a hidden representation with the same process of the standard autoencoder, $\phi = f_\theta(\tilde{x})$.
- From the hidden representation the model reconstructs $z = g_{\theta'}(\phi)$.

The model's parameters θ and θ' are trained to minimize the average reconstruction error over the training data, specifically, minimising the difference between z and the original uncorrupted input x.
Contractive AE (see Rifai et al. [2011]) focus on small variations of the instances around their neighbours by adding a penalty term based on the Frobenius norm of the Jacobian matrix of the activations of the encoder.
Alternatively, variational AE (see Doersch [2016]) introduces regularisation in the regularisation space by encoding data instances using a priori distribution over the latent space, which prevents overfitting and ensures some good properties of the learned space to enable the generation of meaningful data instances. It belongs to the families of

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

variational Bayesian methods. As discussed in Section (12.3.1), given an input dataset x characterised by an unknown probability function $P(x)$ and a multivariate latent encoding vector z, the objective is to model the data as a distribution $p_\theta(x)$, with θ defined as the set of the network parameters, so that

$$\begin{aligned} p_\theta(x) &= \int_z p_\theta(x,z)dz \\ &= \int p(x|\, z;\theta)p(z)dz \end{aligned}$$

where we have used the relationship between the joint distribution and the conditional distribution in Equation (11.2.5). Note, in the autoencoder the latent space is composed of a fixed vector, while in VAE it is composed by a mixture of distributions.

12.4.2.2 Generative adversarial networks

As discussed in Section (12.3.3), generative adversial networks (GANs) aim at learning a latent feature space of the generative network, G, so that it captures well the normality underlying the given data. For example, in the AnoGAN (see Schlegl et al. [2017]), given any data instances x, it aims at searching for an instance z in the learned latent feature space of the generative network G so that the corresponding generated instance $G(z)$ and x are as similar as possible. That is, $\widehat{x} = G(z) \approx x$. It is first trained with the conventional objective (see Equation (12.3.14))

$$\min_G \max_D V(D,G) = E_{x\sim p_\mathcal{X}}[\log D(x)] + E_{z\sim p_\mathcal{Z}}[\log\big(1 - D(G(z))\big)]$$

where G and D are the generator and discriminator networks parametrised by $\mathcal{M}_G$ and $\mathcal{M}_D$, respectively, and V is the value function of the two-player minimax game. The properties of this value function are presented in Section (12.3.3). Recall, we implicitly want to learn the distribution p_G over data x expressed via the generative network G. Then, two loss functions, namely residual loss and discrimination loss are used for each x to find its best z. The former is given by

$$\ell_R(x, z_\gamma) = \|x - G(z_\gamma)\|_1$$

and the latter is given by

$$\ell_{fm}(x, z_\gamma) = \|h(x) - h(G(z_\gamma))\|_1$$

where γ is the index of the search iteration step, and h is a feature mapping from an intermediate layer of the discriminator. The search starts with a randomly sampled z, followed by updating z based on the gradients derived from the overall loss

$$\ell = (1-\alpha)\ell_R(x, z_\gamma) + \alpha\ell_{fm}(x, z_\gamma)$$

where α is a hyperparameter. Note, the parameters of the trained GAN are fixed during this search process, and the loss is only used to update the coefficients of z for the next iteration. Note, this model is computationally inefficient in the iterative search of z. One solution is to add an extra network learning the mapping from data instances into the latent space, that is, an inverse of the generator such as EBGAN (see Zenati et al. [2018a]) or the fast AnoGAN (see Schlegl et al. [2019]).
The EBGAN is built upon the bi-directional GAN (BiGAN) (see Donahue et al. [2017]), which, in addition to the generator G and the discriminator D, has an encoder E mapping x to z in the latent space, and simultaneously learning the parameters of G, D, and E. Rather than discriminating x and $G(z)$, BiGAN aims at discriminating the pair of instances $(x, E(x))$ from the pair $(G(z), z)$. That is,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\min_{G,E}\max_{D} V(D,G) = E_{x\sim p_{\mathcal{X}}}\big[E_{z\sim p_E(\cdot\,|\,x)}[\log D(x,z)]\big] + E_{z\sim p_{\mathcal{Z}}}\big[E_{x\sim p_G(\cdot\,|\,z)}\log\big(1-D(x,z)\big)\big]$$

The anomaly score s_x is given by

$$s_x = (1-\alpha)\ell_G(x) + \alpha\ell_D(x)$$

where

$$\ell_G(x) = \|x - G(E(x))\|_1 \text{ and } \ell_D(x) = \|h(x,E(x)) - h\big(G(E(x)),E(x)\big)\|_1$$

It eliminates the need to iteratively search for z in the AnoGAN. The EBGAN was extended to the ALAD (see Zenati et al. [2018b]) by adding two more discriminators, with one of them trying to discriminate the pair (x,x) from $\big(x,G(E(x))\big)$ and the other one trying to discriminate the pair (z,z) from $\big(z,G(E(z))\big)$. The generator in the previous work was further improved with the GANomaly (see Akcay et al. [2018]), which changed the generator network to an encoder-decoder-encoder network and added two extra loss functions. The generator is represented as

$$x \xrightarrow{G_E} z \xrightarrow{G_D} \widehat{x} \xrightarrow{E} \widehat{z}$$

where G is a composition of the encoder G_E and the decoder G_D. In addition to the commonly used feature matching loss

$$\ell_{fm} = E_{x\sim p_{\mathcal{X}}}[\|h(x) - h(G(x))\|_2]$$

the generator includes a contextual loss

$$\ell_{con} = E_{x\sim p_{\mathcal{X}}}[\|x - G(x)\|_1]$$

and an encoding loss

$$\ell_{enc} = E_{x\sim p_{\mathcal{X}}}[\|G_E(x) - E(G(x))\|_2]$$

to generate realistic instances. The former enforces the generator to consider the contextual information of the input x when generating $\widehat{x}$. The latter helps the generator learning how to encode the features of the generated instances from the training data. The overall loss of the generator is given by

$$\ell = \alpha\ell_{fm} + \beta\ell_{con} + \gamma\ell_{enc}$$

where α, β, and γ are hyperparameters determining the weight of each individual loss. Note, since the training data contains mainly normal instances, the encoders G and E are optimised towards encoding normal instances. When detecting anomalies, one can use the score

$$s_x = \|G_E(x) - E(G(x))\|_1$$

where s_x is expected to be large if x is an anomaly.
A large number of GAN models have been introduced over the years, improving various applications. For instance, in anomaly detection, replacing the standard GAN with the Wasserstein GAN, (see Arjovsky et al. [2017] and Zhu et al. [2017]). Alternatively, one can adversarially learn end-to-end one-class classification models.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

12.4.2.3 Predictability modelling

As discussed in Section (12.6), predictability modelling methods learn representations by predicting the current data instances using the representations of the previous instances within a temporal window as the context. In that setting, data instances are referred to as individual elements in a sequence, such as video frames in a video sequence. It is widely used for sequence representation learning and prediction. In order to obtain accurate predictions, the feature representations are enforced to capture sequential and recurrent dependence within a given sequence length. In general, anomalies violate those dependencies and are unpredictable. Hence, we can use prediction errors such as mean squared errors and likelihood values to define anomaly scores. This approach is popular in video anomaly detection (see Liu et al. [2018b]). For instance, given a video sequence with consecutive t-frames $x_1, x_2, ..., x_t$, the learning task consists in using all these frames to generate a future frame $\widehat{x}_{t+1}$ so that it is as close as possible to the ground truth x_{t+1}. The objective function is given by

$$\alpha \ell_{pred}\big(\widehat{x}_{t+1}, x_{t+1}\big) + \beta \ell_{adv}(\widehat{x}_{t+1})$$

where

$$\widehat{x}_{t+1} = \psi\big(\phi(x_1, x_2, ..., x_t; \mathcal{M}); W\big)$$

ℓ_{pred} is the frame prediction loss measured by mean squared errors, and ℓ_{adv} is the adversarial loss. One can use the U-Net network architecture (see Ronneberger et al. [2015]) to instantiate the function ψ to generate frames. ℓ_{pred} is made of a set of three separate losses respectively enforcing the closeness between $\widehat{x}_{t+1}$ and x_{t+1} in three key image feature descriptors: (1) intensity, (2) gradient, (3) optical flow. ℓ_{adv} is due to the use of adversarial training to enhance the image generation.

A different approach consider autoregressive (AR) models (see Gregor et al. [2014]), which assumes that each element in a sequence is linearly dependent on the previous elements. For instance, Abati et al. [2019] leveraged the AR models to estimate the density of training samples in a latent space, avoiding the assumption of a specific family of distributions. That is, given x and its latent space representation $z = \phi(x; \mathcal{M})$, the AR model factorises the density $p(z)$ as

$$p(z) = \prod_{j=1}^{K} p(z_j | \, z_{1:j-1})$$

where $z_{1:j-1} = \{z_1, z_2, ..., z_{j-1}\}$ and $p(z_j | \, z_{1:j-1})$ represents the probability mass function of z_j conditioned on all the previous instances $z_{1:j-1}$, and K is the dimensionality size of the latent space. The objective being to jointly learn an autoencoder and a density estimation network $\psi(z; W)$ equipped with AR network layers. The overall loss is given by

$$\ell = E_x[\|x - \phi_d((\phi_e(x; \mathcal{M}_e)); \mathcal{M}_d)\|_2 - \lambda \log\left(\psi(z; W)\right)]$$

where the first term is a reconstruction error measured by MSE, while the second term is an AR loss measured by the log-likelihood of the representation under an estimated conditional probability density prior. The minimisation of this loss function enables the learning of the features that are common and easily predictable. In the case of anomaly detection, the anomaly score is obtained by combining the reconstruction error and the log-likelihood at the evaluation stage.

12.4.2.4 Self-supervised classification

Self-supervised classification methods learn representations of normality by identifying instances that are inconsistent to, or disagree, with the classification models of anomalies. Rooted in the cross-feature analysis, or, feature model-based anomaly detection (see Noto et al. [2012]), they evaluate the normality of data instances by their consistency

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(agreement) with a set of predictive models (classification / regression), where each model learns to predict one feature based on the rest of the other features. Rather than using the original data, deep consistency-based anomaly detection focuses on image data and builds the predictive models by using feature transformation-based augmented data (see Golan et al. [2018]). In order to effectively discriminate the transformed instances, the classification models are enforced to learn the features that are highly important to describe the underlying patterns of the instances presented in the training data. As a result, normal instances have stronger agreements with the classification models. In the predictive models, one first apply to a set of normal training images different compositions of geometric feature transformation operations such as horizontal flipping, translations, rotation. A deep multi-class classification model is trained on the augmented data. At the evaluation stage, test instances are augmented with each of the transformation compositions, and their normality score is obtained by aggregating all softmax classification scores to the transformed versions of a given test instance. The loss function is given by

$$\ell_{cons} = CE\big(\psi(z_{T_j}; W); y_{T_j}\big)$$

where $z_{T_j} = \phi(T_j(x); \mathcal{M})$ is a low-dimensional feature representation of instance x augmented by the transformation operator T_j, ψ is a multi-class classifier parametrised with W, y_{T_j} is a one-hot encoding of the synthetic class assigned to instances that are augmented using the transformation operator T_j, and CE is a standard cross-entropy loss function (see Section (2.2)). After minimising the above loss function and getting the optimised classifier ψ^*, we can then apply the feature learner $\phi(\cdot, \mathcal{M}^*)$ and the classifier $\psi(\cdot, W^*)$ to obtain a classification score for each test instance augmented with the operator T_j, which are aggregated to compute the anomaly score. Here, the classification scores conditioned on each operator T_j is assumed to follow a Dirichlet distribution to estimate the consistency of the test instance to the classification model ψ. Alternatively, one can also average the classification scores associated with the operators T_j. Rather than working in a semi-supervised setting, Wang et al. [2019] considered unsupervised setting, with the transformation sets containing four transformation operations: rotation, flipping, shifting, and path-rearranging. In that setting, the gradient magnitude induced by normal instances is larger than the outliers during the training, and the network updating direction is biased towards normal instances. Thus, in general, normal instances have stronger agreement with the classification model than anomalies.

12.4.3 End-to-end one-class classification

End-to-end one-class classification methods aim at training a one-class classifier that learns in an end-to-end manner to discriminate whether a given instance is normal or not. Rather than relying on existing one-class classification measures such as the one-class SVM or SVDD (see Section (11.3)), they directly learn a one-class discrimination model. We combine GANs with the concept of one-class classification, that is, adversarially learned one-class classification. Here, we learn a one-class discriminator of the normal instances so that it discriminates well from some adversarially generated pseudo anomalies. The two main differences with GANs are as follows:

1. GANs aim at learning a generative distribution to maximally approximate the real data distribution, achieving a generative model capturing well the normality of the training instances. On the other hand, end-to-end one-class classification methods aim at optimising a discriminative model to separate training normal instances from adversarially generated fringe instances.

2. GANs define the anomaly scores on the residual between the real instances and the corresponding generated instances. The methods presented in this section directly use the discriminative model to classify anomalies, that is, the discriminator D acts as τ.

It is assumed that all normal instances can be summarised by a discriminative one-class model. Sabokrou et al. [2018] presented the adversarially learned one-class (ALOCC) classification. The main idea being to train two deep networks, with one network trained as the one-class model to separate normal instances from anomalies, while the other network is trained to enhance the normal instances and generate distorted outliers. The two networks are instantiated and

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

optimised in the same way as in the GANs approach. The one-class model is built upon the discriminator network and the generator network is based on denoising AE. The objective function of the AE-empower GAN is given by

$$\min_{AE}\max_{D} V(D,G) = E_{x\sim p_{\mathcal{X}}}[\log D(x)] + E_{\widehat{x}\sim p_{\widehat{\mathcal{X}}}}[\log\left(1 - D(AE(\widehat{x}))\right)]$$

where $p_{\widehat{\mathcal{X}}}$ denotes a data distribution of $\mathcal{X}$ corrupted by a Gaussian noise, that is, $\widehat{x} = x + \eta$ with $\eta \sim N(0, \sigma^2 I)$. This objective function is jointly optimised with the following data construction error in AE:

$$\ell_{ae} = \|x - AE(\widehat{x})\|^2$$

The main idea in the above objective function is that AE can reconstruct well normal instances, but it can be confused by input outliers and, as a result, generates distorted outliers. Hence, through the minmax optimisation, the discriminator D learns to better discriminate normal instances from the outliers than by using the original data instances. Thus, $D(AE(\widehat{x}))$ can be directly used to detect anomalies. Note, the outliers can be randomly drawn from classes other than the classes where the normal instances come from, but obtaining the reference outliers beyond the given training data may be unavailable in several domains. Alternatively, one can generate fringe data instances based on the given training data and use them as negative reference instances to enable the training of the one-class discriminator. For instance, Zheng et al. [2019] introduced one-class adversarial networks (OCAN) to leverage the idea of bad GANs to generate fringe instances based on the distribution of the normal training data. Here, the generator network in bad GANs is trained to generate data instances that are complementary. The objective of the complement generator is given by

$$\min_{G} -\mathcal{H}(p_{\mathcal{Z}}) + E_{\widehat{z}\sim p_{\mathcal{Z}}}[\log p_{\mathcal{X}}(\widehat{z}) I(p_{\mathcal{X}}(\widehat{z}) > \epsilon)] + \|E_{\widehat{z}\sim p_{\mathcal{Z}}}[h(\widehat{z})] - E_{z\sim p_{\mathcal{X}}}[h(z)]\|_2$$

where $\mathcal{H}(\cdot)$ is the entropy, $I(\cdot)$ is the indicator function, ϵ is a threshold hyperparameter, and h is a feature mapping derived from an intermediate layer of the discriminator. The first two terms are devised to generate low-density instances in the original feature space. Since it is computationally infeasible to obtain the probability distribution of the training data, the density estimation $p_{\mathcal{X}}(\widehat{z})$ is approximated by the discriminator of a regular GAN. The last term is the feature matching loss helping better generate data instances within the original data space. The objective of the discriminator in OCAN is enhanced with an extra conditional entropy term to enable the detection with high confidence:

$$\max_{D} E_{x\sim p_{\mathcal{X}}}[\log D(z)] + E_{\widehat{z}\sim p_{\mathcal{Z}}}[\log\left(1 - D(\widehat{z})\right)] + E_{x\sim p_{\mathcal{X}}}[D(x)\log D(x)]$$

Ngo et al. [2019] introduced the Fence GAN with the objective to generate data instances tightly lying at the boundary of the distribution of the training data. The is achieved by introducing two loss functions into the generator that enforce the generated instances to be evenly distributed along a sphere boundary of the training data. The objective of the generator is given by

$$\min_{G} E_{z\sim p_{\mathcal{Z}}}[\log|\alpha - D(G(z))|] + \beta\frac{1}{E_{z\sim p_{\mathcal{Z}}}[\|G(z) - \mu\|_2]}$$

where $\alpha \in (0,1)$ is a hyperparameter used as a discrimination reference score for the generator to generate the boundary instances, and μ is the center of the generated data instances. The first term is the encirclement loss enforcing the generated instances to have the same discrimination score, resulting in instances tightly enclosing the training data. the second term is the dispersion loss enforcing the generated instances to evenly cover the whole boundary. Other methods have been presented to effectively generate the reference instances. Perera et al. [2019] proposed to generate uniformly distributed instances to enforce the normal instances to be distributed uniformly across the latent space.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

12.5 Introduction to complex dynamical systems

12.5.1 Overview

Complex dynamical systems exhibit events on vastly different time scales. As such, they are multifractal. The analysis of such systems include

- Deriving an interpretable model of the long-time dynamical properties of these systems. For instance, the stationary behaviour of cycle times of slow processes.
- Simulating the dynamical system to: (1) predict the future evolution of the system, (2) sample previously unobservable system configurations.

In general, in $[1]$, we assume that the physical systems are Markov, and want to learn a Markov model from the time-series data. In stochastic dynamical systems, such as MD simulation, variants of Markov state models (MSMs) are commonly used (see Prinz et al. [2011], Bowman et al. [2014]). The configuration space is discretised by using clustering methods, and the dynamics between the clusters are described by a matrix of transition probabilities. The analogous approach for deterministic dynamical systems is called Koopman analysis, where time propagation is approximated by a linear model in a suitable function space transformation of the flow variables. For instance, the VAMPnets learn optimal feature transformation from full configuration space to a low-dimensional latent space where the Markov model is built by variational optimisation of neural network (see Mardt et al. [2018]).

In $[2]$, several learning frameworks have been proposed that include a decoder from the latent space back to the space of input features. These methods aim at obtaining a low-dimensional latent space that encodes the long-time behaviour of the system, where the decoder takes the role of defining, or regularising, the learning problem (see Lusch et a. [2017], Ribeiro et al. [2018], Wehmeyer et al. [2018]). However, learning the correct long-time dynamical behaviour with a generative dynamical model is a difficult task. In general, these models did not demonstrate the ability to generate viable structures in the high-dimensional configuration space (in the case of molecular structures). Wu et al. [2018] proposed a deep generative Markov State Model (DeepGenMSM) learning framework for inference of metastable dynamical systems and prediction of trajectories.

12.5.2 Deep generative Markov state models

Given $x, y \in \mathbb{R}^d$, where d can be very large, Markov dynamics are defined by the transition density $P(x_{t+\tau} = y|\, x_t = x)$. Wu et al. [2018] wrote the transition density between m states as (See Figure (12.9))

$$P(x_{t+\tau} = y|\, x_t = x) = \chi^\top(x) q(y;\tau) = \sum_{i=1}^{m} \chi_i(x) q_i(y;\tau)$$

where $\chi^\top(x) = [\chi_1(x), ..., \chi_m(x)]$ represents the probability of configuration x to be in a metastable (long-lived) state i. That is,

$$\chi_i(x) = P(x_t \in \text{ state } i|\, x_t = x)$$

Thus, these functions are non-negative, $\chi_i(x) \geqslant 0\ \forall x$, and sum up to one, $\sum_{i=1}^{m} \chi_i(x)$. Note, the functions $\chi(x)$ can be represented by a neural network mapping from $\mathbb{R}^d$ to $\mathbb{R}^m$ with a softmax output layer. Further, we denote the probability densities

$$q_i(y;\tau) = P(x_{t+\tau} = y|\, x_t \in \text{ state } i)$$

which defines the probability density of the system to arrive at configuration y after making one time-step.
The long-time behaviour of the system is called the kinetics of that system. It is captured by the elements of the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

transition matrix $K = [k_{ij}]$ between metastable states. It is the product of the probability density to jump from state j to a configuration y and the probability that the latter belongs to state j, integrated over the whole configuration space:

$$k_{ij} = \int_y q_i(y;\tau)\chi_j(y)dy$$

Note, the estimated probabilistic functions q and χ define, by construction, a valid transition probability matrix K, that is, $k_{ij} \geqslant 0$ and $\sum_j = k_{ij} = 1$.
The stationary probabilities of the states are given by the vector $\pi = [\pi_i]$ that solves the eigenvalue problem with eigenvalue $\lambda_1 = 1$:

$$\pi = K^\top \pi$$

and the stationary distribution in configuration space is given by

$$\mu(y) = \sum_i \pi_i q_i(y;\tau) = \pi^\top q(y;\tau)$$

For a fixed definition of states via χ, the self-consistency of Markov models may be tested by using the Chapman-Kolmogorov equation

$$K^n(\tau) \approx K(n\tau)$$

which involves estimating the functions $q(y;n\tau)$ at different time lag $n\tau$, and comparing the resulting transition matrices with the nth power of the transition matrix obtained at time lag τ. A consequence of the above equation is that the relaxation times

$$t_i(\tau) = -\frac{\tau}{\log|\lambda_i(\tau)|}$$

are independent of the time lag τ at which K is estimated. Recall, λ_i, $i = 2, ..., m$ are the nontrivial eigenvalues of K.

Wu et al. [2018] proposed a deep generative Markov State Model (DeepGenMSM) that learn the following components:

- Probabilistic encodings of the input configuration to a low-dimensional latent space by neural networks, $x_t \to \chi(x_t)$.
- A true transition probability matrix K describing the system dynamics in latent space for a fixed time-lag τ:

$$E[\chi(x_{t+\tau})] = E[K^\top(\tau)\chi(x_t)]$$

- A generative model from latent vectors back to configurations, allowing to sample the transition density $P(x_{t+\tau}|\, x_t)$ and propagate the model in configuration space.

Given the trajectories $\{x_t; t = 1, ..., T\}$, the authors estimated the probabilities $\chi(x)$ and the sample densities $q(y)$ by assuming that the latter are point densities on the input configurations $\{x_t\}$ given by

$$q_i(y) = \frac{1}{\overline{\gamma}_i}\gamma_i(y)\rho(y)$$

where $\rho(y)$ is the empirical distribution. In the case of finite sample size, it is given by

$$\rho(y) = \frac{1}{T-\tau}\sum_{t=1}^{T-\tau}\delta(y - x_{t+\tau})$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

and $\gamma_i(y)$ is a trainable weighting function. At last, the normalisation factor $\overline{\gamma}_i$ is given by

$$\overline{\gamma}_i = \frac{1}{T-\tau}\sum_{t=1}^{T-\tau}\gamma_i(x_{t+\tau}) = E_{y\sim\rho_1}[\gamma_i(y)]$$

ensures that $\int_y q_i(y)dy = 1$.
Then, χ_i and γ_i are optimised by maximising the likelihood (ML) of generating the pairs $(x_t, x_{t+\tau})$ observed in the data. The log-likelihood is given by

$$l = \sum_{t=1}^{T-\tau}\ln\Big(\sum_{i=1}^{m}\chi_i(x_t)(\overline{\gamma}_i)^{-1}\gamma_i(x_{t+\tau})\Big)$$

It is maximised by training a deep MSM. See Figure (12.9).

Figure 12.9: Deep Generative MSM

The rewiring trick in the above Figure allows to compute the transition matrix by a simple average over all configurations:

$$K = \frac{1}{N}\sum_{t=\tau}^{T-\tau} q(x_{t+\tau})\chi^\top(x_{t+\tau})$$

Note, in the above, the distribution q is derived from empirical distribution, so that any generated trajectory will only resample configurations from the input data. The author called the deep MSM with this methodology DeepResampleMSM. They extended the approach to a model that can generate trajectories that resample from input data as well as produce genuinely new configurations. This is the deep generative MSM (DeepGenMSM). To this end, they trained a generative model to mimic the empirical distribution $q_i(y)$:

$$y = G(e_i, \epsilon)$$

where the vector $e_i \in \mathbb{R}^m$ is a one-hot encoding of the metastable state, and ϵ is a i.i.d. random vector where each component samples from a Gaussian distribution. The generator G is trained by minimising the conditional Energy Distance (ED). It is a metric between the distributions of random vectors given by

$$D_E(P(x), P(y)) = E[2\|x-y\| - \|x-x^{'}\| - \|y-y^{'}\|]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

for two real-valued random variables x and y. Here, $x, x^{'}, y, y^{'}$ are independently distributed according to the distributions of x, y. Based on this metric, the authors introduced the conditional energy distance between the transition density of the system and that of the generative model

$$\begin{aligned} D &= E[D_E(P(x_{t+\tau}|\, x_t), P(\widehat{x}_{t+\tau}|\, x_t))|\, x_t] \\ &= E[2\|x_{t+\tau} - \widehat{x}_{t+\tau}\| - \|x_{t+\tau} - x^{'}_{t+\tau}\| - \|\widehat{x}_{t+\tau} - \widehat{x}^{'}_{t+\tau}\|] \end{aligned}$$

where $x_{t+\tau}$ and $x^{'}_{t+\tau}$ are distributed according to the transition density for given x_t, and $\widehat{x}_{t+\tau}, \widehat{x}^{'}_{t+\tau}$ are independent outputs of the generative model conditioned on x_t. The expectation value is implemented with an empirical average, and G is trained to minimise D. After training, the transition matrix is obtained by using the rewiring trick, where the configuration space integral is sampled by generating samples from the generator

$$[K]_{ij} = E_\epsilon[\chi_j(G(e_i, \epsilon))]$$

The framework was tested on two well defined benchmark systems exhibiting metastable stochastic dynamics: (1) the time discretised diffusion process

$$x_{t+\Delta t} = -\Delta t \nabla V(x_t) + \sqrt{2\Delta t}\dot{\eta}_t$$

with $\Delta t = 0.001$ in the Prinz potential $V(x_t)$, (2) explicit-solvent MD simulations of Alanine dipeptide. In both cases, the authors got approximately correct results for stationary, kinetics and dynamical properties.

12.6 Modelling sequential data

We consider the problem of modelling sequential data of the form $(x_1, ..., x_T) \in \mathcal{X}^T$, for some domain $\mathcal{D}$ and time length T. We assume we have n independent sequence of observations, each having potentially different length, denoted T_i. In that case, each sequence is denoted as $(x_{1,i}, ..., x_{T_i,i}) \in \mathcal{D}^{T_i}$, for $i = 1, ..., n$. For notation simplification, we sometimes use $x_{\leqslant t} = x_1, x_2, ..., x_t$ or $x_{1:t}$.

12.6.1 RNNs as statistical models

12.6.1.1 An Overview

As discussed in Section (12.1.2), Learning Networks can be derived from the classical regularisation problem and are good candidates for predicting non-linear time series. For instance, the feed-forward multilayer networks were expressed as a regression model in Section (12.1.2.2). Recurrent neural networks (RNNs) (see Section (6.1)) are a rich class of dynamic models that can be trained for sequence generation by processing real data sequences one step at a time and making one-step ahead prediction. Graves [2013] considered a deep recurrent neural network (RNN) composed of stacked LSTM layers, and explained how this network could be trained for the next-step prediction, and hence sequence generation. Sutskever et al. [2014] proposed a general sequence to sequence learning framework where a recurrent network is used to encode a sequence into a fixed length representation, and then another recurrent network is used to decode a sequence out of that representation. Assuming the predictions are probabilistic, we can generate new sequences from the trained network by iteratively sampling from the network's output distribution which are fed as input at the next step. Further, in the presence of unsupervised sequential data, RNNs can model the generative process of the sequential data by learning to estimate the probability distribution of the next item in the sequence given all the previous ones.
As a results, RNNs have been used to generate sequences in various domains such as music (see Boulanger-Lewandowski et al. [2012]), text (see Sutskever et al. [2011]), speech recognition (see Graves et al. [2014b]), motion capture data (see Sutskever et al. [2008]), on videos for recognising actions and generating natural language descriptions (see Donahue et al. [2014]).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We also saw in Section (6.3) that Reservoir Computing models were expressed in terms of state-space models (see Appendix (23.4.3)). Recently, some authors established the connections between general RNNs and statistical models such as state-space models (SSMs), which include hidden Markov models (HMMs) (see Baum et al. [1966]) and linear Gaussian SSMs or the Kalman filtering model (see Kalman [1960]). SSMs are a class of probabilistic graphical models that encode the dependency structure of sequential data by latent variables that follow Markovian dynamics. For instance, Mohamed [2015] interpreted a simple RNN as the maximum likelihood estimate of a state-space model following deterministic dynamics. That is, RNNs can be viewed as a sophisticated version of SSMs with highly non-linear transition function. In standard probabilistic models, the inference procedure is separated from the model architecture (see Section (11.2)). However, in the case of neural networks, Andreas [2016] showed that SMMs build a model-inference pair that could not be easily identified separately. In the case of recurrent neural networks, he noted that a hidden unit could not be viewed as encoding a single hidden state as in a hidden Markov model, but that it encoded the whole table of probabilities of the HMM forward algorithm. Choe et al. [2017] considered the representation of RNNs in terms of graphical models. They also took a functional approximation view point as well as an approximate inference view point, to formalise the connection between RNNs and HMMs.

12.6.1.2 RNNs as state-space models

We briefly introduced state-space models (SSMs) in Appendix (23.4.3) and discussed their parameter estimation in Section (11.2.5). We change notation and follow the approach presented by Choe et al. [2017]. Being at time t, the input variable (or variable source) is denoted by x_t, the state variable is denoted by h_t, and the output is the next input value $y_t = x_{t+1}$, for every $t = 1, ..., T-1$. As discussed in Section (6.1), assuming a single layer, an RNN predicting the next item in the sequence can be defined in terms of a state space model (SSM) as

$$\begin{aligned} h_s &= f_\alpha(h_{s-1}, x_s)\,, \forall s = 1, ..., t \\ x_{t+1} &= g_\beta(h_t) \end{aligned} \tag{12.6.16}$$

for some non-linear functions f_α and g_β parametrised by vector-valued parameters α and β, respectively. Here, the functions are time independent. Graph theory (see Section (13.1)) can be used to represent state space models. The computational graph of the model is given in Figure (12.10).

Figure 12.10: Computational graph of a RNN.

As an example, we can define a simple RNN model as follows:

$$\begin{aligned} f_\alpha(h_{s-1}, x_s) &= \mathcal{H}_f(Wx_s + Uh_{s-1} + b)\,, \forall s = 1, ..., t \\ g_\beta(h_t) &= \mathcal{H}_g(Vh_t + c) \end{aligned} \tag{12.6.17}$$

where $\alpha = (W, U, b)$ and $\beta = (V, c)$ for some weight matrices W, U, V and bias vectors b, c. The weight matrices are filters determining how much importance should be given to both the present input and the past hidden state. Note, U

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

is the transition matrix. The functions $\mathcal{H}_f$ and $\mathcal{H}_g$ are non-linear activation functions. Thus, RNNs are a generalisation of Markov chains, where the hidden states are supposedly encoding all previous history in the sequence.

***Remark* 12.6.1** *This representation corresponds to the formulation of the Reservoir Computing model (see Section (6.3)) given in Equations (6.3.9) and (6.3.10), respectively, with no feedback matrix.*

In order to probabilistically model sequential data, one must assume that the sequential data come from a larger population, and that it is subject to an element of randomness. Thus, we must therefore rely on statistical inferences and the notion of random sample (see Section (11.2)). In order to introduce the notion of probability when modelling sequential data, we first need to introduce the random variables $X_1, ..., X_T$ defined on the domain $\mathcal{D}$ to model the observations $x_1, ..., x_T$. We also need to introduce the latent variables $H_1, ..., H_T$ to model the hidden states corresponding to each item in the sequence. In that setting, dataset is produced by a randomisation design. The computational graph of the model is given in Figure (12.11).

Figure 12.11: Computational graph of state-space model.

We use probabilistic inference to estimate unobserved values. To do so, we need to estimate the target distribution

$$P(H_1, ..., H_T | \, x_1, ..., x_T)$$

This is the full posterior distribution of the hidden states given the observation measurement process.

12.6.1.3 The particle filter

Particle filters, or Sequential Monte Carlo (SMC) methods, are a set of Monte Carlo algorithms used to solve filtering problems arising in signal processing and Bayesian statistical inference. The filtering problem consists of estimating the internal states in dynamical systems when partial observations are made, and random perturbations are present in the sensors as well as in the dynamical system. The objective is to compute the posterior distributions of the states of some Markov process, given some noisy and partial observations.
Particle filtering uses a set of particles (also called samples) to represent the posterior distribution of some stochastic process given noisy and/or partial observations. The state-space model can be non-linear and the initial state and noise distributions can take any form required. Particle filter techniques provide a well-established methodology for generating samples from the required distribution without requiring assumptions about the state-space model or the state distributions. However, these methods do not perform well when applied to very high-dimensional systems.

A generic particle filter estimates the posterior distribution of the hidden states using the observation measurement process. We consider the state-space model shown in Figure (12.11). The filtering problem is to estimate sequentially the values of the hidden states H_T given the values of the observation process $x_1, ..., x_T$ at any time step T. All Bayesian estimates of H_T follow from the posterior density $P(H_T \, | x_1, ..., x_T)$. The particle filter methodology provides an approximation of these conditional probabilities using the empirical measure associated with a genetic

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

type particle algorithm. In contrast, the MCMC, or Importance sampling approach, would model the full posterior distribution $P(H_1, ..., H_T \mid x_1, ..., x_T)$.

In the case of the above non-linear state-space model, we apply the particle filter, which is a stochastic inference strategy. It is a sequential importance sampling algorithm (see Appendix (26.2.2.4)) with intermediate re-sampling steps. It works by approximating the target distribution

$$P(H_1, ..., H_T | \, x_1, ..., x_T)$$

in an on-line method [9] by using a set of M weighted samples

$$\big(H_{1,1}, ..., H_{1,T}, W_1\big) \cdots \big(H_{M,1}, ..., H_{M,T}, W_M\big)$$

where $W = \{W_1, ..., W_M\}$ is a set of weight matrices. We let $q(H_t | \, H_{t-1})$ denote a proposal distribution equal to, or similar to, $P(H_t | \, H_{t-1})$. Let H_0 be arbitrary. For a state space model, the particle filter algorithm is given by

1. Propagate the particles $H_{i,t} \sim q(H_t | \, H_{t-1})$, $i = 1, ..., M$.
2. Compute the weights
$$W_{i,t} = \frac{P(H_t | \, x_t, H_{t-1})}{q(H_t | \, H_{t-1})}$$
3. Normalise the weights $\sum_{i=1}^{M} W_{i,t} = 1$.
4. Re-sample to obtain M equally weighted particles
$$\big(H_{1,1}, ..., H_{1,t}, \frac{1}{M}\big) \cdots \big(H_{M,1}, ..., H_{M,t}, \frac{1}{M}\big)$$

Iterating the steps for $t = 1, ..., T$ results in the approximation of the target distribution

$$P(H_1, ..., H_T | \, x_1, ..., x_T) \approx \sum_{i=1}^{M} W_{i,t} \delta_{H_{i,1:t}}$$

where δ_{ij} is the Kronecker delta (see Appendix (23.2.1.5)).

12.6.1.4 Link with generative models

We follow the approach proposed by Choe et al. [2017] and describe how RNNs can be represented in terms of graphical models (see Section (13.2)).
Given binary random sequences $(X_1, ..., X_T) \in \{0, 1\}^T$ with time length T, as well as deterministic functions f_α and g_β (sigmoidal functions) in Equation (12.6.16), a RNN induces a set of steps to model the conditional distribution of X_{t+1} given $X_{\leqslant t}$, for all $t = 1, ..., T$. For example, to model the conditional probability $P(X_{t+1} = 1 | \, X_{\leqslant t} = x_{\leqslant t})$ (since X_t takes value in $\{0, 1\}$):

1. Calculate a set of inner states $\{h_s; s = 0, ..., t\}$ which is recursively defined by
$$h_0 = 0 \, , h_s = f_\alpha(h_{s-1}, x_s) \; \forall s = 1, ..., t$$

[9] In computer science, an online algorithm is one that can process its input piece-by-piece in a serial fashion, that is, in the order that the input is fed to the algorithm, without having the entire input available from the start.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. The conditional probability is given by

$$P(X_{t+1} = 1 | \, X_{\leqslant t} = x_{\leqslant t}; \alpha, \beta) = g_\beta(h_t) \tag{12.6.18}$$

where h_t depends on $x_1, ..., x_t$ and α. Thus, given the functions f_α and g_β, the output of the RNN at each time step $t+1$ is the conditional probability of X_{t+1} given $X_{\leqslant t}$.

Using RNN, from the conditional density in Equation (24.1.16) with $k = 1$, we model the conditional distribution of $(X_2, ..., X_T)$ given X_1 as follows:

$$P(X_2, ..., X_T | \, X_1) = \prod_{t=1}^{T-1} P(X_{t+1} | X_{\leqslant t})$$

Thus, given n training sequences $\{x_{i,1}, ..., x_{i,T_i}; i = 1, ..., n\}$, we estimate the parameters (α, β) by using the maximum conditional likelihood estimator

$$\begin{aligned}(\widehat{\alpha}, \widehat{\beta}) &= \arg\max_{\alpha,\beta} \sum_{i=1}^{n} \log P\big(x_{i,2}, ..., x_{i,T} | x_{i,1}; \alpha, \beta\big) \\ &= \arg\max_{\alpha,\beta} \sum_{i=1}^{n} \sum_{t=1}^{T_i - 1} \log P\big(x_{i,t+1} | x_{i,\leqslant t}; \alpha, \beta\big)\end{aligned}$$

Note, the RNN does not model the marginal distribution of X_1, so that we need to consider another model for $P(X_1)$ in order to model the full joint distribution of $(X_1, ..., X_T)$. Given that formulation of a RNN, we can derive a graphical model interpretation based on the graph G given in Figure (12.12).

Figure 12.12: Computational graph G of RNN model.

The structure of the graph G induces the conditional independence relationships (see Section (13.2.2.2))

$$\begin{aligned}&X_{t+1} \perp X_{\leqslant t} | \, H_t \; \forall t = 1, ..., T-1 \\ &X_t \perp X_{\leqslant t-1} | \, H_{t-1}, X_t \; \forall t = 2, ..., T-1\end{aligned}$$

It states that the future observation is independent to the current and past observations given the current hidden state, and the current hidden state is independent to past observations given the hast hidden state and current observation. For given RNN parameters (α, β), we can define a graphical model $M(G; \alpha, \beta)$ with the graph structure G and the parametrisation

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} P(X_{t+1} = 1 | \, H_t = h_t) &= g_\beta(h_t) \\ P(H_t | \, H_{t-1} = h_{t-1}, X_t = x_t) &= \delta(H_t = f_\alpha(h_{t-1}, x_t)) \\ P(H_1 | \, X_1 = x_1) &= \delta(H_1 = f_\alpha(0, x_1)) \end{aligned}$$

Here, the graphical model parametrisation do not assume a deterministic relationship between the random variables $\{H_t; t = 1, ..., T\}$ and $\{X_t; t = 1, ..., T\}$. The authors showed that these parametrisations induce an additional conditional independence relationship. This is expressed in the following proposition.

***Proposition* 3** *Under the graphical model parametrisation, the graphical model $M(G; \alpha, \beta)$ has the additional property*

$$H_t \perp X_{\geqslant t+1} | \, H_{t-1}, X_t \; \forall t$$

It implies that the current hidden state is not only independent to the past observations, but also to the future observations, given the past hidden state and current observation. A consequence of this proposition is that the graphical model $M(G; \alpha, \beta)$ is equivalent to the RNN in the generative model view point.

Theorem 12.6.1 *The graphical model $M(G; \alpha, \beta)$ defined above induces the same conditional distribution of $(X_2, ..., X_T)$ given X_1 as one induced by the corresponding RNN.*

Thus, RNNs can be interpreted as probabilistic graphical models that involve deterministic transition probabilities between the hidden variables.

12.6.2 Generating sequences

12.6.2.1 Description

When applying machine learning to sequences (see Appendix (23.2.1.7)), we may want to turn an input sequence into an output sequence that lives in a different domain. As an example, we consider the sentence pair $(x, y) \in (\mathcal{X}, \mathcal{Y})$, where $\mathcal{X}$ and $\mathcal{Y}$ are the source and target domain, respectively. When there is no separate target sequence, we can get a teaching signal by trying to predict the next term in the input sequence. That is, the target output sequence is the input sequence with an advance of 1 step. For temporal sequences there is a natural order for the predictions.
However, predicting the next term in a sequence makes it difficult to distinguish between supervised and unsupervised learning. This is because it uses methods designed for supervised learning, but it does not require a separate teaching signal.

As discussed in Section (12.1.1), we can use memoryless models for predicting sequences such as autoregressive moving average (ARMA) models (see Equation (12.1.4)), or feedforward neural nets (MLP) (see Section (5.1.3)), which generalise AR models by using one or more layers of non-linear hidden units. We can also let the generative models have some hidden states with their own internal dynamics, such as state-space models (SSMs) (see Section (11.2.5)). As a result, we obtain richer models for predicting sequences because we can store long term information in those hidden states. However, in the case of noisy data and noisy outputs generated by the hidden states, we can not observe the true hidden states. Thus, in that environment, we can only infer a probability distribution over the space of hidden state vectors (see Section (11.2)). This inference is only tractable for certain types of hidden state models. Some of these models are:

- Linear Dynamical Systems (LDS): It corresponds to generative models having a non-observable real-valued hidden state. The hidden state has linear dynamics with Gaussian noise and produces the observations using a linear model with Gaussian noise. They are also called Gaussian models. Thus, we first need to infer the hidden

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

state to predict the next output. Since a linearly transformed Gaussian variable is also Gaussian, the distribution at time t, over the hidden state, given the data up to t, is Gaussian. Hence, it can be computed with Kalman filters (see Section (11.2.5.2)).

- Hidden Markov Models (HMMs): They have a discrete one-of-N hidden state and the transitions between states are stochastic and controlled by a transition matrix. The outputs produced by a state are stochastic. We can represent a probability distribution across N states with N numbers. Inferring the probability distribution over the hidden states with some algorithms, we can then predict the next output. When generating data, at each time step the algorithm must select one of its hidden states. However, with N hidden states it can only remember $\log(N)$ bits about what it generated so far. For example, the information that the first half of an utterance must contain about the second half can be of the order of 100 bits, leading to 2^{100} which is enormous.

Even though both linear dynamical systems (LDS) and hidden Markov models (HMMs) are stochastic models, the posterior probability distribution over their hidden states, given the observed data up to time t, is a deterministic function of the data. This was demonstrated by Choe et al. [2017] and presented in Section (12.6.1.4). Since recurrent neural networks (RNNs) are deterministic, we can think of the hidden state of an RNN as the equivalent of the deterministic probability distribution over hidden states in a LDS or HMM.

12.6.2.2 Generating sequences with RNNs

12.6.2.2.1 An overview As discussed in Section (12.6.1), recurrent neutral networks (RNNs) are the primary deep learning method for sequential data. They can be trained for sequence generation by processing real data sequences one step at a time and predicting what comes next. RNNs can model the generative process (see Section (11.2.4)) of the sequential data by learning to estimate the probability distribution of the next item in the sequence given all the previous ones. Assuming the predictions are probabilistic, we can generate new sequences by iteratively sampling from the output distribution of a trained network. The sample is fed as input at the next step. The obtained distribution is conditional, since the internal state of the network, and hence its predictive distribution, depends on the previous inputs. Predictions is made by using the RNNs internal representation to perform a high-dimensional interpolation between training examples. That is, RNNs synthesise and reconstitute the training data in a complex way. As such, they do not use exact templates from the training data to make predictions.

In the sequence generation problem, we saw in Section (6.1) that recurrent neural networks (RNNs) have difficulties in modelling long sequences as they are unable to store information about past inputs for very long, and that extensions have been proposed to handle this problem. When the recurrent networks cannot make sense of their recent history, they need longer memory to look back in the past to formulate the predictions. LSTM is an example of such extension that is better at storing and accessing information than standard RNNs. We presented in Section (6.1.5) a mathematical representation of RNNs including the LSTM, the GRU, and reservoir computing as special cases. As a variant of deep RNNs, Graves [2013] proposed Stacked RNNs which build up extra layers of hidden units that accept the hidden units of the previous layer as their input sequences.

12.6.2.2.2 The stacked LSTM model Graves [2013] considered a deep recurrent neural network (RNN) composed of stacked LSTM layers, and explained how this network could be trained for the next-step prediction, and hence sequence generation. Following Section (12.6.1), we represent the recurrent neural network as a state-space model (SSM). At time t, we get the input x_t, three hidden layers $h_{n,t}$, $n = 1, 2, 3$ and one output y_t. This is similar to the unfolded RNN in time displayed in Figure (6.2), except that the output y_t becomes the input x_{t+1} at time $t + 1$. The RNN prediction architecture is illustrated in Figure (12.13). In that setting, the output y_t is used to parameterise the predictive distribution $P(x_{t+1} \mid y_t)$ (since we are computing the next-step prediction of the input sequence).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 12.13: The circles mean network layers, the solid lines mean weighted connections and the dashed lines mean predictions.

Given an input vector sequence $x = (x_1, ..., x_T)$ with time length T, a stack of N recurrently connected hidden layers ($N = 1$ is a single layer), the hidden vector sequences $h_n = (h_{n,1}, ..., h_{n,T})$, $n = 1, ..., N$, we compute the output vector sequence $y = (y_1, ..., y_T)$. Each output vector y_t is used to parameterise a predictive distribution $P(x_{t+1} \mid y_t)$ over the possible next inputs x_{t+1}. Letting the first element x_1 to be null, the network emits a prediction for x_2 with no prior information. There are some skip connections from the inputs to all hidden layers, and from all hidden layers to the outputs. It makes it easier to train deep networks by reducing the number of processing steps between the bottom of the network and the top, thus mitigating the vanishing gradient problem. The hidden layer activations are computed by iterating equations from $t = 1$ to T and from $n = 2$ to N. These equations (see Section (6.1) and Equation (12.6.17)) are

$$\begin{aligned} h_{1,t} &= f\big(W_{ih_1}x_t + W_{h_1h_1}h_{1,t-1} + b_{1,h}\big) \\ h_{n,t} &= f\big(W_{ih_n}x_t + W_{h_{n-1}h_n}h_{n-1,t} + W_{h_nh_n}h_{n,t-1} + b_{n,h}\big) \end{aligned} \tag{12.6.19}$$

where W_{ih_n} is the weight matrix connecting the inputs to the nth hidden layer, $W_{h_1h_1}$ is the recurrent connection at the first hidden layer, and so on. Given the hidden sequences, the output sequence is given by

$$\begin{aligned} \widehat{y}_t &= b_y + \sum_{n=1}^{N} W_{h_ny}h_{n,t} \\ y_t &= f_{out}(\widehat{y}_t) \end{aligned} \tag{12.6.20}$$

with a stack of N recurrently connected hidden layers.
The complete network defines a function, parameterised by the weight matrices, from input histories $x_{1:t}$ to output vectors y_t.

Statement 12.6.1 *The output vectors are used to parameterise the predictive distribution $P(x_{t+1} \mid y_t)$ for the next input. The form of $P(x_{t+1} \mid y_t)$ must be chosen carefully to match the input data.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

***Remark* 12.6.2** *However, finding a good predictive distribution for high-dimensional, real-valued data (referred to as density modelling), can be very challenging.*

From the conditional density in Equation (24.1.16), the probability given by the network to the input vector sequence $x = (x_1, ..., x_T)$ is

$$P(x) = \prod_{t=1}^{T} P(x_{t+1} \mid y_t)$$

and the sequence loss $\mathcal{L}(x)$ used to train the network is the negative logarithm of the probability $P(x)$

$$\mathcal{L}(x) = -\sum_{t=1}^{T} \log P(x_{t+1} \mid y_t)$$

This loss function can be differentiated with respect to the network weights with backpropagation through time applied to the computation graph, so that the network can be trained with gradient descent. We can use this learned probability distribution to sample a new sequence.

12.6.2.2.3 Mathematical representation Being at time t, the input variable (or variable source) is denoted by x_t, the state variable is denoted by h_t, and the output is the next input value $y_t = x_{t+1}$, for every $t = 1, ..., T-1$. Graves [2013] proposed the fully connected LSTM (FC-LSTM), which is a multivariate version of LSTM where the input, cell output and states are all 1D vectors (same size as the hidden vector h). In addition to the representation in Section (6.1.5), each gate has an internal source, the cell state c_{t-1} of its cell block. The FC-LSTM architecture is defined as follows:

$$f_{LSTM}(c_{t-1}, h_{t-1}, x_t)$$

where

$$\begin{aligned}
i_t &= f_\sigma\big(W_i x_t + U_i h_{t-1} + V_i c_{t-1} + b_i\big) \\
f_t &= f_\sigma\big(W_f x_t + U_f h_{t-1} + V_f c_{t-1} + b_f\big) \\
\tilde{c}_t &= g\big(W_{\tilde{c}} x_t + U_{\tilde{c}} h_{t-1} + b_c\big) \\
c_t &= i_t \circ \tilde{c}_t + f_t \circ c_{t-1} \\
o_t &= f_\sigma\big(W_o x_t + U_o h_{t-1} + V_o c_t + b_0\big) \\
h_t &= o_t \circ g(c_t)
\end{aligned}$$

where f_σ is a sigmoid function and $g = \tanh$ is a hyperbolic tangent activation function. The operator $\circ$ denotes the Hadamard product [10]. Also, $\alpha = (W, U, V)$ for some weight matrices W, U, and V. Note, all the $V_{c\cdot}$ matrices are diagonal, whereas the rest are dense. The full gradient can be calculated with backpropagation through time (see Graves et al. [2005]).

[10] Given two matrices A and B, the Hadamard product is the entry-wise multiplication of two matrices, which makes sense when both matrices have the same number of rows and columns. It is defined as

$$(A \circ B)[i, j] = A[i, j]B[i, j]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

12.6.3 Reconstructing and predicting sequences

12.6.3.1 The encoder-decoder-predictor model

Srivastava et al. [2015c] proposed an encoder-decoder approach (see Section (14.1)) to learn representations of video sequences in the unsupervised setting. They used an encoder LSTM to map an input sequence into a fixed length representation which is decoded by using single or multiple decoder LSTMs producing a target sequence. This is used to perform different tasks, such as reconstructing the input sequence, or predicting the future sequence. They showed that the representations help improve classification accuracy. They considered two types of inputs: image patches, and high-level percepts extracted by applying a convolutional net trained on ImageNet. The outputs were quantitatively analysed. They proposed a composite model combining the two above tasks and showed that it performed best.

12.6.3.1.1 The autoencoder model Autoencoder models learn some low-dimensional feature representation space on which the given data instances can be well reconstructed (see Section (12.4.2)). Here, the model consists of two RNNs: the encoder LSTM and the decoder LSTM. This is illustrated in Figure (12.14). The encoder reads in the input sequence and the decoder outputs a prediction for the target sequence. The target sequence is same as the input sequence, but in reverse order to ease the optimisation. While the encoder creates a list, the decoder unrolls this list, with the hidden to output weights extracting the element at the top of the list and the hidden to hidden weights extracting the rest of the list.

Figure 12.14: LSTM autoencoder.

The decoder can be of two kinds:

1. Conditional: It receives the last generated output frame (or variable) as input, that is, the dotted input in Figure (12.14) is present. It allows the decoder to model multiple modes in the target sequence distribution. When predicting future values of a sequence, there is a possibility of multiple targets given an input. However, it might be a problem when the sequence exhibit strong short-range correlations.

2. Unconditioned: It does not receive that input. This input is removed and the model is forced to look for information deep inside the encoder.

The architecture can be extend to multiple layers by stacking LSTMs on top of each other. Note, the state of the encoder LSTM after the last input has been read is the representation of the input (not an identity mapping) containing its characteristics. Two factors control this behaviour:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1. A fixed number of hidden units: It makes it unlikely that the model can learn trivial mappings for arbitrary length input sequences.
2. The same LSTM operation is used to decode the representation recursively, meaning that the same dynamics must be applied on the representation at any stage of decoding.

12.6.3.1.2 The predictor model Another natural unsupervised learning task for the sequences is predicting their future values. The predictor model is same as that of the autoencoder model above, except that the decoder LSTM in this case predicts frames (or variables) that come after the input sequence. This is illustrated in Figure (12.15). The hidden state from the encoder capture the dynamics of the representation of the input sequence, while the decoder can be conditional or unconditioned (as above).

Figure 12.15: LSTM predictor.

12.6.3.1.3 The composite model A high-capacity autoencoder suffer from the tendency to learn trivial representations that just memorise the inputs. However, this memorisation is not useful at all for predicting the future variables. This is because when predicting v_t, the variables $\{v_{t-1}, ..., v_{t-k}\}$ are far more important than the variables $\{v_{t-k-1}, ..., v_{t-k-l}\}$ for $l > k$. That is, the representation at the end of the encoder have a tendency to be forgotten. One solution is to combine the two tasks of reconstructing the input and predicting their future values to create a composite model. This is illustrated in Figure (12.16). The encoder LSTM produces a state used both to reconstruct the input and predict the next few variables.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 12.16: LSTM composite.

12.6.3.2 Convolutional LSTM network

The stacked LSTM encoder-decoder framework described in Section (12.6.2.2) provides a general framework for sequence-to-sequence learning problems by training temporally concatenated LSTMs, one for the input sequence and another for the output sequence. This is detailed in Section (14.1). For instance, Ranzato et al. [2014] showed that the prediction of the next video frame and interpolation of intermediate frames could be done by building an RNN based language model on the visual words obtained by quantizing the image patches. While they proposed a recurrent convolutional neural network to model the spatial relationships, the model only predicts one frame ahead and the size of the convolutional kernel used for state-to-state transition is restricted to one. Focusing on multi-step prediction, Srivastava et al. [2015c] built an LSTM encoder-decoder-predictor model which reconstructs the input sequence and predicts the future sequence simultaneously (see Section (12.6.3.1)). However, the fully connected LSTM (FC-LSTM) layer adopted by their model does not take spatial correlation into consideration. In order to locally predict the future rainfall intensity, Shi et al. [2015] formulated precipitation nowcasting as a spatiotemporal sequence forecasting problem in which both the input and the prediction target are spatiotemporal sequences. They proposed the convolutional LSTM (ConvLSTM) by extending the fully connected LSTM (FC-LSTM) to have convolutional structures in both the input-to-state and state-to-state transitions. They showed that this network captures spatiotemporal correlations better and consistently outperformed the FC-LSTM.

12.6.3.2.1 Mathematical formulation We let $\mathcal{X}$ and $\mathcal{Y}$ be the input and output spaces, $\mathcal{D}$ the probability distribution, D the data set, H the set of hypotheses, $I(\cdot)$ the indicator function. We let $\mathscr{X} \in \mathcal{X}$ be the input tensor, and $\mathscr{Y} \in \mathcal{Y}$ be the output tensor. We consider the spatiotemporal sequence forecasting problem where we observe a dynamical

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

system over a spatial region represented by an $M \times N$ grid which consists of M rows and N columns. Inside each cell in the grid, there are P measurements which vary over time, so that the observation at any time can be represented by a tensor $\mathscr{X} \in \mathbb{R}^{P \times M \times N}$. This is illustrated in Figure (12.17).

Figure 12.17: Transforming 2D image into 3D tensor.

Recording the observations periodically, at time $t \in [1, T]$, the input variables are denoted by $\mathscr{X}_t, \mathscr{X}_{t-1}, ..., \mathscr{X}_{t-J+1}$ and the outputs are the future input values $\mathscr{Y}_{t+1} = \mathscr{X}_{t+1}, ..., \mathscr{Y}_{t+K} = \mathscr{X}_{t+K}$, $K > 0$, for every $t = 1, ..., T$. The spatiotemporal sequence forecasting problem consists in predicting the most likely length-K sequence in the future given the previous J observations including the current one. This is an arg max problem given by

$$\widehat{\mathscr{Y}}_{t+1}, ..., \widehat{\mathscr{Y}}_{t+K} = \arg \max_{\mathscr{Y}_{t+1}, ..., \mathscr{Y}_{t+K}} P\big(\mathscr{Y}_{t+1}, ..., \mathscr{Y}_{t+K} | \, \mathscr{X}_t, ..., \mathscr{X}_{t-J+2}, \mathscr{X}_{t-J+1}\big)$$

Here, the prediction target of the forecasting problem is a sequence which contains both spatial and temporal structures. The number of free variables in a length-K sequence can be as large as $O(M^K N^K P^K)$. We can exploit the structure of the space of possible predictions to reduce the dimensionality and hence make the problem tractable.
In the case of the precipitation nowcasting problem, the observation at every timestamp is a 2D radar echo map which can be formulated as a spatiotemporal sequence forecasting problem by dividing the map into tiled non-overlapping patches and viewing the pixels inside a patch as its measurements.

12.6.3.2.2 The model Shi et al. [2015] stacked multiple ConvLSTM layers and formed an encoding-forecasting structure, getting a network model for general spatiotemporal sequence forecasting problems. Here, all the inputs $\mathscr{X}_1, ..., \mathscr{X}_t$, the cell outputs $\mathscr{C}_1, ..., \mathscr{C}_t$, the hidden states $\mathscr{H}_1, ..., \mathscr{H}_t$, and the gates i_t, f_t, o_t of the ConvLSTM are 3D tensors whose last two dimensions are spatial dimensions (rows and columns). Similarly to the LSTM, the ConvLSTM determines the future state of a certain cell in the grid by the inputs and past states of its local neighbours. This done by using a convolution operator in the state-to-state and input-to-state transitions. The is illustrated in Figure (12.18).

Figure 12.18: Inner structure of ConvLSTM cell.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The key equations of ConvLSTM are as follows:

$$f_{ConvLSTM}(\mathscr{C}_{t-1}, \mathscr{H}_{t-1}, \mathscr{X}_t)$$

where

$$\begin{aligned}
i_t &= f_\sigma\big(W_i * \mathscr{X}_t + U_i * \mathscr{H}_{t-1} + V_i \circ \mathscr{C}_{t-1} + b_i\big) \\
f_t &= f_\sigma\big(W_f * \mathscr{X}_t + U_f * \mathscr{F}_{t-1} + V_f \circ \mathscr{C}_{t-1} + b_f\big) \\
\tilde{\mathscr{C}_t} &= g\big(W_{\tilde{c}} * \mathscr{X}_t + U_{\tilde{c}} * \mathscr{H}_{t-1} + b_c\big) \\
\mathscr{C}_t &= i_t \circ \tilde{\mathscr{C}_t} + f_t \circ \mathscr{C}_{t-1} \\
o_t &= f_\sigma\big(W_o * \mathscr{X}_t + U_o * \mathscr{H}_{t-1} + V_o \circ \mathscr{C}_t\big) \\
\mathscr{H}_t &= o_t \circ g(\mathscr{C}_t)
\end{aligned}$$

where f_σ is a sigmoid function and $g = \tanh$ is a hyperbolic tangent activation function. Here, $*$ denotes the convolution operator and the operator $\circ$ denotes the Hadamard product [11]. Also, $\alpha = (W, U, V)$ for some weight matrices W, U, and V. The forget gate decides what to keep from the previous cell memory in the component-wise product operation $(f_t \circ \mathscr{C}_{t-1})$, while the input gate controls what candidate values are to be used in the $(i_t \circ \tilde{\mathscr{C}_t})$ operation, similarly to the forget gate. By combining the result of these two operations the current cell state is formed. Finally the hidden state $\mathscr{H}_t$ results in cell state that was previously calculated above, passed through a hyperbolic tangent activation function and filtered by the output gate. These gating mechanisms make sure that dependencies are captured across long time ranges. The full gradient can be calculated with backpropagation through time (see Graves et al. [2005]).

Similarly to FC-LSTM, the ConvLSTM can also be adopted as a building block for more complex structures. As an example, the authors considered a structure with two networks, an encoding network and a forecasting network. Both networks are formed by stacking several ConvLSTM layers. This is illustrated in Figure (12.19).

Figure 12.19: Encoding-forecasting ConvLSTM network.

Following Srivastava et al. [2015c] (see Section (12.6.3.1)), the initial states and cell outputs of the forecasting network are copied from the last state of the encoding network. Since the prediction target has the same dimensionality as the input, the authors concatenated all the states in the forecasting network and fed them into a 1×1 convolutional layer to generate the final prediction. Here, the encoding LSTM compresses the whole input sequence into a hidden

[11] Given two matrices A and B, the Hadamard product is the entry-wise multiplication of two matrices, which makes sense when both matrices have the same number of rows and columns. It is defined as

$$(A \circ B)[i, j] = A[i, j]B[i, j]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

state tensor $\mathscr{H}$ and the forecasting LSTM unfolds this hidden state to give the final prediction. Mathematically, we get

$$\mathscr{Z}_t = f_E(\mathscr{X}_t, ..., \mathscr{X}_{t-n+2}, \mathscr{X}_{t-n+1})$$

and

$$\begin{aligned}
\widehat{\mathscr{Y}}_{t+1}, ..., \widehat{\mathscr{Y}}_{t+m} &= \arg \max_{\mathscr{Y}_{t+1},...,\mathscr{Y}_{t+m}} P\big(\mathscr{Y}_{t+1}, ..., \mathscr{Y}_{t+n} | \, \mathscr{X}_t, ..., \mathscr{X}_{t-n+2}, \mathscr{X}_{t-n+1}\big) \\
&\approx \arg \max_{\mathscr{Y}_{t+1},...,\mathscr{Y}_{t+m}} P\big(\mathscr{Y}_{t+1}, ..., \mathscr{Y}_{t+m} | \, \mathscr{Z}_t\big) \\
&\approx g_F\big(\mathscr{Z}_t\big)
\end{aligned}$$

where f_E is the encoding function and g_F is the forecasting function (see Section (12.6.3.1)). Here, the input and output elements are all 3D tensors, which preserve all the spatial information. The network has multiple stacked ConvLSTM layers leading to strong representational power suitable for predictions in complex dynamical systems.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 13

Graph networks

13.1 Graph theory

See books by Biggs et al. [1986], Bondy et al. [2008], Bang-Jensen et al. [2010], among other.

13.1.1 Description

Graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. Many practical problems can be represented by graphs. It lies at the heart of combinatorial optimisation. Emphasising their application to real-world systems, the term network is sometimes defined to mean a graph in which attributes (such as names) are associated with the vertices and edges. The concept of graph dynamical systems (GDS) can be used to capture a wide range of processes taking place on graphs or networks. A major theme in the mathematical and computational analysis of graph dynamical systems is to relate their structural properties (such as the network connectivity) and the global dynamics that result.

13.1.1.1 An overview

In his seminal article , Euler [1736] introduced the concept of graph theory where he studied the relation between the number of edges, vertices, and faces of a convex polyhedron. This work was generalised by Cauchy and L'Huilier, who laid the foundation of topology (see Biggs et al. [1986]). Later, Cayley [1857] studied a particular class of graphs, the tree, by considering the enumeration of graphs with particular properties. He linked his results on trees with contemporary studies of chemical composition, resulting in the standard terminology of graph theory. Sylvester used the term graph to draw an analogy between "quantic invariants" and "co-variants" of algebra and molecular diagrams. Textbooks on the subject were published, such as the ones by Konig, Haray [1969], among others.
Guthrie proposed the four color problem: "Is it true that any map drawn in the plane may have its regions coloured with four colors, in such a way that any two regions having a common border have different colors?" This problem remained unsolved for more than a century. In 1969, Heinrich Heesch [1969] published a method for solving the problem using computers and the notion of "discharging". The proof involved checking the properties of $1,936$ configurations.
Another important factor of common development of graph theory and topology came from the use of the techniques of modern algebra. The first example of such a use comes from the work of the physicist Gustav Kirchhoff, who published in 1845 his Kirchhoff's circuit laws for calculating the voltage and current in electric circuits.
The introduction of probabilistic methods in graph theory, especially in the study of Erdos and Renyi of the asymptotic probability of graph connectivity, gave rise to yet another branch, known as random graph theory, which has been a fruitful source of graph-theoretic results. Since networks with a complex topology and unknown organising principles appeared random, they have been studied with random graph.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

13.1.1.2 Graph concepts

A graph is a structure amounting to a set of objects in which some pairs of the objects are in some sense related. The objects correspond to mathematical abstractions called vertices (also called nodes or points) and each of the related pairs of vertices is called an edge (also called an arc, or link, or line). Typically, a graph is depicted in diagrammatic form as a set of dots for the vertices, joined by lines or curves for the edges.
An undirected graph (or just graph) $G = (V, E)$ consists of a node set V and an edge set E, which are two-element subsets of V. That is, $E \subseteq \{[x, y] \mid (x, y) \in V^2 \wedge x \neq y\}$. A node (or vertex) can be any object. An edge is a pair of nodes, taking the form of an unordered pair comprising those two nodes. It is equivalent to saying that an edge joins or connects two nodes. Other senses of graph stem from different conceptions of the edge set. In one of them, V is a set together with a relation of incidence that associates two vertices with each edge. In another generalised notion, E is a multiset of unordered pairs of (not necessarily distinct) vertices. Many authors call this type of object a multigraph or pseudograph.
More generally, allowing for multiple edges, a graph is an ordered triple $G = (V, E, \phi)$ comprising V a set of vertices, E a set of edges, and $\phi : E \to \{[x, y] \mid (x, y) \in V^2 \wedge x \neq y\}$ an incidence function mapping every edge to an unordered pair of vertices. This is an undirected multigraph.

Graphs can be used to model various structures and relations. The vertices belonging to an edge are called the ends or end vertices of the edge. V and E are usually taken to be finite. In general, we denote the vertices by numbers $1, ..., N$ (sometimes $v_1, ..., v_n$) and the edge between vertex i and vertex j by $[i, j]$ or $i - j$. The edge $e = [i, j] \in E$ is uniquely identified by its endpoints i and j. The edges have no direction, so that $[i, j] = [j, i]$.
Vertices $i \in V$ and $j \in V$ are adjacent (or neighbours) if $[i, j] \in E$ (they are the endpoints of an edge in G). Then, we say that edge $e = [i, j]$ is incident to i and j. The order of a graph is $|V|$, its number of vertices. The size of a graph is $|E|$, its number of edges. The degree, or valency, of a vertex is the number of edges incident to it. In an undirected simple graph of order n, the maximum degree of each vertex is $n - 1$ and the maximum size of the graph is $N = \binom{n}{2} = \frac{n(n-1)}{2}$ [1].
In a weighted graph, each edge $[i, j]$ has a weight w_{ij} associated with it. Some examples of weights are distances, costs or capacities.

Graphs are represented visually by drawing a point or circle for every vertex, and drawing a line between two vertices if they are connected by an edge. If the graph is directed, the direction is indicated by drawing an edge.
For example, we illustrate in Figure (13.1) the node set
$V = \{1, 2, 3, 4, 5, 6\}$ and the edge set $\{[1, 2], [1, 3], [2, 3], [2, 5], [2, 6], [3, 4], [4, 5], [5, 6]\}$.

Figure 13.1: Graph example

[1] The combination symbol $C_r^n = \frac{n!}{(n-r)!r!}$ is also called the binomial coefficient $\binom{n}{r}$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

13.1.1.2.1 Some concepts We now list a few concepts around graphs:

- A directed graph (or digraph) $D = (V, A)$ consists of a node (or vertex) set V and an arc set A (directed edge set), where an arc is an ordered pair of nodes (it is a directed edge). That is, $A \subseteq \{(x, y) \mid (x, y) \in V^2 \wedge x \neq y\}$ is a set of directed edges. Arc from node i to node j is denoted by (i, j) or $i \to j$, where i is the tail of the edge and j is its head. The edge (j, i) is called the inverted edge of (i, j). In diagrams, arcs are often depicted as arrows. A path is a sequence of edges having the property that the ending vertex of each edge in the sequence is the same as the starting vertex of the next edge in the sequence. A path forms a cycle if the starting vertex of its first edge equals the ending vertex of its last edge. See Figure (13.2). A mixed graph has both directed and undirected edges. Loops are arrows that connect nodes with themselves, that is, (i, i) or $i \to i$. Directed graphs without loops are addressed as simple directed graphs, while directed graphs with loops are addressed as loop-digraphs (a binary relation between a set and itself). A directed simple graph with loops $G = (V, A)$ is symmetric if, for every edge in A, the corresponding inverted edge also belongs to A.

Figure 13.2: Euler directed graph

- A subgraph or subdigraph of a given graph or digraph $G = (V, E)$, respectively, is a graph or digraph obtained by deleting some of the edges and vertices of G, retaining the other edges of G (together with their pairs of endpoints).
- A walk in a graph $G = (V, E)$ is a sequence of adjacent nodes, denoted by $w = [v_1, v_2, ..., v_k]$ such that $[v_i, v_{i+1}] \in E$ for $i = 1, ..., k-1$. It can also be thought of as a sequence of edges $[v_1, v_2], [v_2, v_3], ..., [v_{k-1}, v_k]$. The walk is closed if $v_1 = v_k (k > 1)$. A closed walk is called a circuit. A walk with no repeated nodes is a path. A walk $w = [v_1, v_2, ..., v_k]$ with no repeated node except for $v_1 = v_k$ is a cycle. Directed walk, path and cycle are defined accordingly, using notation $w = (v_1, v_2, ..., v_k)$.
- An acyclic graph is a graph having no graph cycles (see above definition of a cycle). Acyclic graphs are bipartite. A connected acyclic graph is called a tree, and a possibly disconnected acyclic graph is known as a forest (a collection of trees).
- A directed acyclic graph (DAG), is a finite directed graph with no directed cycles. That is, it consists of finitely many vertices and edges (also called arcs), with each edge directed from one vertex to another, such that there is no way to start at any vertex v and follow a consistently-directed sequence of edges that eventually loops back to v again.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- A graph is complete if there is an edge between every pair of vertices, that is, if all vertices are adjacent to each other. It is connected if there is a path between every pair of vertices. A graph is planar if it can be drawn in the plane without crossing edges.
- A tree is a connected graph with no cycles.
- A Hamiltonian path is a path that includes every vertex exactly once. A Hamiltonian cycle is a cycle that includes every vertex exactly once. It is a closed tour through edges of E that visits every node of V exactly once. The length (cost) of a Hamiltonian cycle is the sum of lengths of all traversed edges.
- An Eulerian walk / circuit is a walk / closed walk, in which each edge appears exactly once.
- A circulant graph is an undirected graph acted on by a cyclic group of symmetries which takes any vertex to any other vertex. Every cycle graph is a circulant graph, as is every crown graph with two modulo four vertices.
- A bipartite graph $G = (V, E)$ is a graph in which the vertex set V is partitioned into sets S and T (without common elements) such that every edge of G has one end in S and the other in T, so that there are no edges in G that have both ends in S or both ends in T. Such a graph is also written $G = (S, T; E)$.

A vertex v of a directed graph is said to be reachable from another vertex u when there exists a path that starts at u and ends at v. If a vertex can reach itself via a non-trivial path (a path with one or more edges), then that path is a cycle.

13.1.1.2.2 Distance The distance between two vertices in a graph is the number of edges in a shortest path (see Section (13.1.2.1)) connecting them. It is also known as the geodesic distance. In the case of a directed graph, the distance $d(u, v)$ between two vertices u and v is defined as the length of a shortest directed path from u to v consisting of arcs, provided that at least one such path exists. Contrary to the case of undirected graphs, $d(u, v)$ does not necessarily coincide with $d(v, u)$, and it might be the case that one is defined while the other is not.
A metric space defined over a set of points in terms of distances in a graph defined over the set is called a graph metric. The vertex set of an undirected graph and the distance function form a metric space if and only if the graph is connected. The eccentricity $\epsilon(v)$ of a vertex v is the greatest distance between v and any other nodes. It is denoted by $\epsilon(v) = \max_{u \in V} d(v, u)$. It tells us how far a node is from the node most distant from it in the graph. The radius r of a graph is the minimum eccentricity of any vertex given by

$$r = \min_{v \in V} \epsilon(v) = \min_{v \in V} \max_{u \in V} d(v, u)$$

The diameter d of a graph is the maximum eccentricity of any vertex in the graph. That is, d is the greatest distance between any pair of vertices given by

$$d = \max_{v \in V} \epsilon(v)$$

To find the diameter of a graph we first find the shortest path between each pair of vertices. Then, the greatest length of any of these paths is the diameter of the graph. Put another way, the diameter of a graph is the longest distance (number of hops) between any two vertices. We use the notation Θ to mean on the order of.
A central vertex in a graph of radius r is one whose eccentricity is r, that is, a vertex that achieve the radius, or equivalently, a vertex v such that $\epsilon(v) = r$. A pseudo-peripheral vertex v has the property that for any vertex u, if v is as far away from u as possible, then u is as far away from v as possible. That is, a vertex u is pseudo-peripheral, if for each vertex v with $d(u, v) = \epsilon(u)$ holds $\epsilon(u) = \epsilon(v)$. A graph such that for every pair of vertices there is a unique shortest path connecting them is called a geodetic graph. As an example, all trees are geodetic.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

13.1.1.2.3 Clique A clique is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent, that is, its induced subgraph is complete. Formally, a clique C in an undirected graph $G = (V, E)$ is a subset of the vertices $C \subseteq V$, such that every two distinct vertices are adjacent. It is equivalent to the condition that the induced subgraph of G induced by C is a complete graph. A maximal clique is a a clique that cannot be extended by including one more adjacent vertex, that is, a clique which does not exist exclusively within the vertex set of a larger clique. A maximum clique of a graph G is a clique such that there is no clique with more vertices. Further, the clique number $\omega(G)$ of a graph G is the number of vertices in a maximum clique in G. The intersection number of G is the smallest number of cliques that together cover all edges of G. The clique cover number of a graph G is the smallest number of cliques of G whose union covers the set of vertices V of the graph. A maximum clique transversal of a graph is a subset of vertices with the property that each maximum clique of the graph contains at least one vertex in the subset. The opposite of a clique is an independent set. The clique cover problem concerns finding as few cliques as possible that include every vertex in the graph.

13.1.1.3 Storage

There are different ways for storing graphs in a computer system. The data structure used depends on both the graph structure and the algorithm used for manipulating the graph. Theoretically one can distinguish between the following two structures:

1. List structures: They include the incidence list, an array of pairs of vertices, and the adjacency list, which separately lists the neighbours of each vertex. Much like the incidence list, each vertex has a list of which vertices it is adjacent to. List structures are often preferred for sparse graphs as they have smaller memory requirements.

2. Matrix structures: They include the incidence matrix, a matrix of 0's and 1's whose rows represent vertices and whose columns represent edges, and the adjacency matrix, in which both the rows and columns are indexed by vertices. In both cases a 1 indicates two adjacent objects and a 0 indicates two non-adjacent objects. The distance matrix, like the adjacency matrix, has both its rows and columns indexed by vertices, but rather than containing a 0 or a 1 in each cell it contains the length of a shortest path between two vertices. Matrix structures provide faster access for some applications but can consume huge amounts of memory.

The adjacency matrix $A = [a_{ij}]$ of a graph G with

$$a_{ij} = \begin{cases} 1 \text{ if G has a (directed) edge } (i,j) \\ 0 \text{ otherwise} \end{cases}$$

is symmetric. The vertex incidence list of a graph shows for each vertex the incident edges. The edge incident list shows for each edge its two endpoints. Similarly for a digraph. In the vertex list, outgoing edges then get a minus sign, and in the edge list we have ordered pairs of vertices. In general, the adjacency matrix of a graph is much smaller than the incidence matrix. However, in concrete applications the best structure is often a combination of both.
Sparse graphs are graphs with few edges (far less than $N = \frac{n(n-1)}{2}$, where n is the number of vertices). For these graphs, matrices are less efficient than lists that require much less storage and are easier to handle.

13.1.1.4 Hierarchical structure

A hierarchy is an organisational form consisting of an arrangement of items (objects, names, values, categories, among others) in which the items are represented as being above, below, or at the same level as one another. The direct links in a hierarchy are the ones immediately superior and the ones subordinates.
The Degree of branching refers to the number of direct subordinates or children an object has (in graph theory, equivalent to the number of other vertices connected to via outgoing arcs, in a directed graph) a node has. Hierarchies can be categorised based on the maximum degree: the highest degree present in the system as a whole. It leads to three broad classes:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1. Linear hierarchy: The maximum degree is 1. All of the objects can be visualised in a line-up, and each object (excluding the top and bottom ones) has exactly one direct subordinate and one direct superior.
2. Branching hierarchy: One or more objects has a degree of 2 or more (and therefore the minimum degree is 2 or higher). Branching hierarchies are present within numerous systems, including organisations and classification schemes. The broad category of branching hierarchies can be further subdivided based on the degree.
3. Flat hierarchy: It is a branching hierarchy in which the maximum degree approaches infinity. For example, diamonds and graphite are flat hierarchies of numerous carbon atoms which can be further decomposed into subatomic particles.

In graph networks, the degree of a node is the number of connections (or edges) it has to other nodes and the degree distribution is the probability distribution of these degrees over the whole network. If a network is directed (edges point in one direction from one node to another node) then nodes have two different degrees, the in-degree, which is the number of incoming edges, and the out-degree, which is the number of outgoing edges. The degree distribution $P(k)$ is the fraction of nodes in the network with degree k. Thus, if there are n nodes in total in a network and n_k of them have degree k, we have $P(k) = \frac{n_k}{n}$. We can also consider a cumulative degree distribution, which is the fraction of nodes with degree smaller than k, $P(X \leqslant k)$. The degree distribution is very important in studying both real networks, such as the Internet and social networks, and theoretical networks.

13.1.1.5 Some simple graphs

13.1.1.5.1 Regular graphs A regular graph is a graph where each vertex has the same number of neighbours. That is, every vertex has the same degree or valency. A regular directed graph must also satisfy the stronger condition that the indegree and outdegree of each vertex are equal to each other. A regular graph with vertices of degree k is called a k-regular graph. Regular graphs of degree at most two are easy to classify: a 0-regular graph consists of disconnected vertices, a 1-regular graph consists of disconnected edges, and a 2-regular graph consists of a disjoint union of cycles and infinite chains.
A strong regular graph is a regular graph where every adjacent pair of vertices has the same number of neighbours in common, and every non-adjacent pair of vertices has the same number n of neighbours in common. The smallest regular graphs, but not strongly regular are the cycle graph and the circulant graph on six vertices. Further, every k-regular graph on $2k+1$ vertices has a Hamiltonian cycle. The necessary and sufficient conditions for a k-regular graph of order n to exist are that $n \geqslant k+1$ and that nk is even. In such case, one can construct regular graphs by considering appropriate parameters for circulant graphs.
Let A be the adjacency matrix of a graph. Then the graph is regular if and only if $J = (1, ..., 1)$ is an eigenvector of A. Its eigenvalue will be the constant degree of the graph. Eigenvectors corresponding to other eigenvalues are orthogonal to J. Thus, for such eigenvectors $v = (v_1, ..., v_n)$, we have $\sum_{i=1}^{n} v_i = 0$. A regular graph of degree k is connected if and only if the eigenvalue k has multiplicity one. Let G be a k-regular graph with diameter D and eigenvalues of adjacency matrix $k = \lambda_0 > \lambda_1 \geqslant \cdots \geqslant \lambda_{n-1}$. If G is not bipartite, then

$$D \leqslant \frac{\log(n-1)}{\log \frac{\lambda_0}{\lambda_1}} + 1$$

13.1.1.5.2 Complete graphs A complete graph is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge. A complete digraph is a directed graph in which every pair of distinct vertices is connected by a pair of unique edges. The complete graph on n vertices is denoted by K_n. It has $\frac{n(n-1)}{2}$ edges (a triangular number), and it is a regular graph of degree $n-1$. All complete graphs are their own maximal cliques. They are maximally connected as the only vertex cut which disconnects the graph is the complete set of vertices. The complement graph of a complete graph is an empty graph. If the edges of a complete graph are each given an orientation, the resulting directed graph is called a tournament. The number of matchings of the complete graphs are given by the telephone numbers. The number of perfect matchings of the complete graph K_n (with n even) is given

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

by the double factorial $(n-1)!!$.
A complete graph with n nodes represents the edges of an $(n-1)$-simplex. For instance, K_3 forms the edge set of a triangle, K_4 a tetrahedron, and so on. K_1 through K_4 are all planar graphs. A graph is planar if and only if it contains neither K_5 nor the complete bipartite graph $K_{3,3}$ as a subdivision.

13.1.2 Examples of combinatorial problems

So far we have described basic concepts and notations needed in graph theory. We now consider some important classes of combinatorial problems.

13.1.2.1 Shortest path problems

The shortest path problem is the problem of finding a path between two vertices in a graph such that the sum of the weights of its constituent edges is minimised. This problem can be defined for undirected, directed, or mixed graphs.

13.1.2.1.1 Description In a graph $G = (V, E)$ we can walk from a vertex v_1 along some edges to some other vertex v_k. There are three related concepts:

1. Walk: we can make no restrictions, so that a walk from v_1 to v_k is of the form

$$(v_1, v_2), (v_2, v_3), \cdots, (v_{k-1}, v_k)$$

   where some of these edges or vertices may be the same.

2. Trail: we require that each edge of G be traversed at most once.

3. Path: we require that each vertex be visited at most once.

All of the above is closed if its last vertex coincides with the first one, that is, $v_k = v_1$. A closed path is called a cycle. Since we do not permit double edges, a cycle has at least three edges.
We consider a graph $G = (V, E)$ in which each edge (v_i, v_j) has a given length $l_{ij} > 0$. We want to find the shortest path $v_1 \to v_k$, that is, a path such that the sum of the lengths of its edges

$$l_{12} + l_{23} + \cdots + l_{k-1,k}$$

is minimum. Similarly, a longest path $v_1 \to v_k$ is one for which that sum is maximum. Note, the length l_{ij} is also called cost or weight. For example, the travelling salesman problem requires the determination of a shortest Hamiltonian cycle in a given graph, that is, a cycle that contains all the vertices of the graph. Further, by choosing the most profitable route $v_1 \to v_k$, a salesman may want to maximise $\sum l_{ij}$, where l_{ij} is his expected commission minus his travel expenses for going from town i to town j.

13.1.2.1.2 Bellman's optimality principle We write $l_{ij} = \infty$ for any edge (i, j) that does not exist in G. We denote vertex 1 the origin, and $2, 3, ..., n$ all the other vertices of G. We let L_j denote the length of a shortest path $1 \to j$ in G.

Definition 13.1.1 *Bellman's minimality principle*
If $P : 1 \to j$ is a shortest path from 1 to j in G and (i, j) is the last edge of P, then $P_i : 1 \to i$ (obtained by dropping (i, j) from P) is a shortest path $1 \to i$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Proof: Suppose that the conclusion is false. Then there is a path $P_i^* : 1 \to i$ that is shorter than P_i. Hence, if we now add (i, j) to P_i^*, we get a path $1 \to j$ that is shorter than P. It contradicts the assumption that P is shortest.
From the Bellman's principle, we can derive basic equations as follows: For fixed j we may obtain various paths $1 \to j$ by taking shortest paths P_i for various i and adding to them an edge (i, j). These paths $1 \to j$ have lengths $L_i + l_{ij}$, where L_i is the length of P_i. We can now take the minimum over i, that is, pick an i for which $L_i + l_{ij}$ is smallest. By the Bellman's principle, it gives a shortest path $1 \to j$. It has the length

$$L_1 = 0 \text{ and } L_j = \min_{i \neq j}\big(L_i + l_{ij}\big) \text{ for } j = 2, ..., n$$

These are the Bellman equations. Since $l_{ii} = 0$ by definition, we can rewrite $\min_{i \neq j}$ with $\min_i$.

13.1.2.1.3 Shortest spanning trees A tree T is a graph that is connected [2] and has no cycles. A spanning tree T in a given connected graph $G = (V, E)$ is a tree containing all the n vertices of G. Such a tree has $n - 1$ edges. A shortest spanning tree T in a connected graph G whose edges (i, j) have length $l_{ij} > 0$ is a spanning tree for which $\sum l_{ij}$ (sum over all edges of T) is minimum compared to $\sum l_{ij}$ for any other spanning tree in G. As an example, consider a private communication network G, let p_{ij} be the probability of intercepting line (i, j) by an outsider, and suppose that one wants to communicate a confidential message to all participants (vertices) along a spanning tree T in G that minimises the product of the p_{ij} of all the edges (lines) of T, that is, assuming stochastic independence, the total probability of interception. Equivalently, a tree T that minimises the sum of the logarithms of the p_{ij}, or, the sum of $l_{ij} = \log \tilde{p}_{ij}$, where $\tilde{p}_{ij} = Kp_{ij}$ with K so large that $\tilde{p}_{ij} > 1$ for each edge of the network G, thus $l_{ij} > 0$.

13.1.2.2 Networks

A network is a digraph $G = (V, E)$ in which each edge (i, j) has assigned to it a capacity $c_{ij} > 0$ (maximum possible flow along (i, j)), and at one vertex s (called the source) a flow is produced that flows along the edges to another vertex t (called the target or sink), where the flow disappears. It may be the flow of electricity in wires, of water in pipes, of cars on road, of people in public transportation system, and so on. The two basic tools for networks are flow augmenting paths and cut sets.

13.1.2.2.1 Flow augmenting paths We denote the flow along a directed edge (i, j) by f_{ij} and impose two conditions:

1. Edge condition: For each edge (i, j) in G, $0 \leqslant f_{ij} \leqslant c_{ij}$.
2. Vertex condition: For each vertex, not s or t, Inflow $=$ Outflow .

In a formula

$$\sum_k f_{ki} - \sum_j f_{ij} = \begin{cases} 0 \text{ if vertex } i \neq s\,, i \neq t \\ -f \text{ at the source } s \\ f \text{ at the target } t \end{cases}$$

Here, f is the total flow. Given the definition of a path above, when we travel from v_1 to v_k, we may travel through some edge in its given direction, which we call forward edge, or opposite to its given direction, which we call backward edge. In order to maximise the flow f in a network, we look for methods of increasing an existing flow. We want to find a path $P : s \to t$ whose edges are not yet fully used, so that we can push additional flow through P by

1. increasing the flow on forward edges, and
2. decreasing the flow on backward edges.

[2] Connected means that there is a path from any vertex in T to any other vertex in T.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We get the following definition:

Definition 13.1.2 *A flow augmenting path in a network with a given flow f_{ij} on each edge (i,j) is a path $P : s \to t$ such that*

1. *no forward edge is used to capacity; thus $f_{ij} < c_{ij}$ for these.*
2. *no backward edge has flow 0; thus $f_{ij} > 0$ for these.*

13.1.2.2.2 Cut sets A cut set is a set of edges in a network. If we want to find what is flowing from s to t in a network, we can cut the network somewhere between s and t and see what is flowing in the edges hit by the cut, since any flow from s to t must pass through some of these edges. It forms a cut set denoted by (S,T), where S is the set of vertices on that side of the cut on which s lies and T is the set of the other vertices. We say that a cut partition the vertex set V into two parts S and T. The corresponding cut set (S,T) consists of all the edges in the network with one end in S and the other end in T.
By definition, the capacity cap (S,T) of a cut set (S,T) is the sum of the capacities of all forward edges in (S,T) (forward edges only), that is, the edges that are directed from S to T

$$cap(S,T) = \sum c_{ij}$$

The other edges (directed from T to S) are called backward edges of the cut set (S,T), and by the net flow through a cut set we mean the sum of the flows in the forward edges minus the sum of the flows in the backward edges of the cut set.

Theorem 13.1.1 *Net flow in cut set*
Any given flow in a network G is the net flow through any cut set (S,T) of G.

One consequence of this theorem is the following:

Theorem 13.1.2 *Upper bound for flows*
A flow f in a network G cannot exceed the capacity of any cut set (S,T) in G.

Cut sets bring out the full importance of augmenting paths:

Theorem 13.1.3 *Augmenting path theorem for flows*
A flow from s to t in a network G is maximum if and only if there does not exist a flow augmenting path $s \to t$ in G.

Another important result is as follows:

Theorem 13.1.4 *Max-flow min-cut theorem*
The maximum flow in any network G equals the capacity of a minimum cut set (a cut set of minimum capacity) in G.

13.1.3 Directed acyclic graph

A directed acyclic graph (DAG) is a directed graph that has a topological ordering, a sequence of the vertices such that every edge is directed from earlier to later in the sequence. That is, it is a directed graph with no cycles (see Bang-Jensen et al. [2010]). It is illustrated in Figure (13.3). The corresponding concept for undirected graphs is a forest, an undirected graph without cycles. Choosing an orientation for a forest produces a special kind of directed acyclic graph called a polytree. However, there are many other kinds of directed acyclic graph that are not formed by orienting the edges of an undirected acyclic graph. Moreover, every undirected graph has an acyclic orientation, an assignment of a direction for its edges that makes it into a directed acyclic graph. To emphasize that DAGs are not the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

same thing as directed versions of undirected acyclic graphs, some authors call them acyclic directed graphs or acyclic digraphs.

Figure 13.3: Topological ordering of a DAG

DAGs can represent collections of events and their influence on each other, either in a probabilistic structure such as a Bayesian network or as a record of historical data such as family trees or the version histories of distributed revision control systems. DAGs can also be used as a compact representation of sequence data, such as the directed acyclic word graph representation of a collection of strings, or the binary decision diagram representation of sequences of binary choices. More abstractly, the reachability relation in a DAG forms a partial order, and any finite partial order may be represented by a DAG using reachability.

13.1.3.1 Some properties

We present a few mathematical properties:

- Reachability: The reachability relationship in any directed acyclic graph can be formalised as a partial order $\leqslant$ on the vertices of the DAG. In this partial order, two vertices u and v are ordered as $u \leqslant v$ exactly when there exists a directed path from u to v in the DAG; that is, when v is reachable from u. However, different DAGs may give rise to the same reachability relation and the same partial order. For instance, the DAG with two edges $a \to b$ and $b \to c$ has the same reachability relation than the graph with three edges $a \to b$, $b \to c$, and $a \to c$. Both of these DAGS produce the same partial order, in which the vertices are ordered as $a \leqslant b \leqslant c$.

- Transitive closure: If G is a DAG, its transitive closure is the graph with the most edges that represents the same reachability relation. It has an edge $u \to v$ whenever u can reach v. That is, it has an edge for every related pair $u \leqslant v$ of distinct elements in the reachability relation of G, and may therefore be thought of as a direct translation of the reachability relation $\leqslant$ into graph-theoretic terms. The same method of translating partial orders into DAGs works more generally: for every finite partially ordered set $(S, \leqslant)$, the graph that has a vertex for each member of S and an edge for each pair of elements related by $u \leqslant v$ is automatically a transitively

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

closed DAG, and has $(S, \leqslant)$ as its reachability relation. In this way, every finite partially ordered set can be represented as the reachability relation of a DAG.

Figure 13.4: A DAG G and its transitive closure.

- Transitive reduction: The transitive reduction of a DAG G is the graph with the fewest edges that represents the same reachability relation as G. It is a subgraph of G, formed by discarding the edges $u \to v$ for which G also contains a longer path connecting the same two vertices. See Figure (13.5). Similarly to the transitive closure, the transitive reduction is uniquely defined for DAGs. In contrast, for a directed graph that is not acyclic, there can be more than one minimal subgraph with the same reachability relation. Transitive reductions are useful in visualizing the partial orders they represent, because they have fewer edges than other graphs representing the same orders and therefore lead to simpler graph drawings. A Hasse diagram of a partial order is a drawing of the transitive reduction in which the orientation of each edge is shown by placing the starting vertex of the edge in a lower position than its ending vertex.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 13.5: A DAG G and its transitive reduction.

- topological ordering: Every DAG has a topological ordering, an ordering of the vertices such that the starting endpoint of every edge occurs earlier in the ordering than the ending endpoint of the edge. The existence of such an ordering can be used to characterise DAGs: a directed graph is a DAG if and only if it has a topological ordering. In general, this ordering is not unique; a DAG has a unique topological ordering if and only if it has a directed path containing all the vertices, in which case the ordering is the same as the order in which the vertices appear in the path.

13.1.4 Topological ordering and sorting

See textbook by Cormen et al. [2001].

13.1.4.1 Topological ordering

A topological ordering of a directed graph is an ordering of its vertices into a sequence, such that for every edge the start vertex of the edge occurs earlier in the sequence than the ending vertex of the edge. A graph that has a topological ordering cannot have any cycles, because the edge into the earliest vertex of a cycle would have to be oriented the wrong way. Therefore, every graph with a topological ordering is acyclic. Conversely, every directed acyclic graph has at least one topological ordering. Therefore, this property can be used as an alternative definition of the directed acyclic graphs: they are exactly the graphs that have topological orderings. The family of topological orderings of a DAG is the same as the family of linear extensions of the reachability relation for the DAG, so any two graphs representing the same partial order have the same set of topological orders.

13.1.4.2 Topological sorting

Topological sorting is the algorithmic problem of finding a topological ordering of a given DAG. It is based on a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering. As an example, the vertices of a graph may represent tasks to be performed, and the edges may represent constraints that one task must be performed before another one. In general, the algorithm can be solved in linear time, that is, linear in the number of nodes plus the number of edges. We get asymptotically $O(|V| + |E|)$. For instance, Kahn's algorithm for topological sorting builds the vertex ordering directly (see Kahn [1962]). It maintains a list of vertices that have no incoming edges from other vertices that have not already been included in the partially constructed

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

topological ordering; initially this list consists of the vertices with no incoming edges at all. Then, it repeatedly adds one vertex from this list to the end of the partially constructed topological ordering, and checks whether its neighbours should be added to the list. The algorithm terminates when all vertices have been processed in this way. Alternatively, a topological ordering may be constructed by reversing a postorder numbering of a depth-first search graph traversal.

13.1.5 Causal structures

Graphs in which vertices represent events occurring at a definite time, and where the edges are always point from the early time vertex to a late time vertex of the edge, are necessarily directed and acyclic (DAG). The lack of a cycle follows because the time associated with a vertex always increases as you follow any path in the graph so you can never return to a vertex on a path. This reflects our natural intuition that causality means events can only affect the future, they never affect the past, and thus we have no causal loops. An example of this type of directed acyclic graph are those encountered in the causal set approach to quantum gravity though in this case the graphs considered are transitively complete. The version history example, each version of the software is associated with a unique time, typically the time the version was saved, committed or released. For citation graphs, the documents are published at one time and can only refer to older documents.
Sometimes events are not associated with a specific physical time. Provided that pairs of events have a purely causal relationship, that is edges represent causal relations between the events, we will have a directed acyclic graph (DAG). For instance, a Bayesian network represents a system of probabilistic events as vertices in a directed acyclic graph, in which the likelihood of an event may be calculated from the likelihoods of its predecessors in the DAG. In this context, the moral graph of a DAG is the undirected graph created by adding an (undirected) edge between all parents of the same vertex (sometimes called marrying), and then replacing all directed edges by undirected edges. Another type of graph with a similar causal structure is an influence diagram, the vertices of which represent either decisions to be made or unknown information, and the edges of which represent causal influences from one vertex to another. In epidemiology, for instance, these diagrams are often used to estimate the expected value of different choices for intervention.

13.1.6 Some applications

Graphs can be used to model many types of relations and processes in physical, biological, social and information systems. Many practical problems can be represented by graphs. The term network is sometimes defined to mean a graph in which attributes (such as names) are associated with the vertices and edges, and the subject that expresses and understands the real-world systems as a network is called network science.
Some applications are as follows:

- Computer science: Graphs are used to represent networks of communication, data organisation, computational devices, the flow of computation, and many more. For instance, the link structure of a website can be represented by a directed graph, in which the vertices represent web pages and directed edges represent links from one page to another. A similar approach can be taken to problems in social media, travel, biology, computer chip design, mapping the progression of neuro-degenerative diseases, and many other fields. The development of algorithms to handle graphs is therefore of major interest in computer science. The transformation of graphs is often formalized and represented by graph rewrite systems. Complementary to graph transformation systems focusing on rule-based in-memory manipulation of graphs are graph databases geared towards transaction-safe, persistent storing and querying of graph-structured data.

- Linguistics: Graph-theoretic methods, in various forms, have proven particularly useful in linguistics, since natural language (NL) often lends itself well to discrete structure. Traditionally, syntax and compositional semantics follow tree-based structures, whose expressive power lies in the principle of compositionality, modelled in a hierarchical graph. More contemporary approaches such as head-driven phrase structure grammar model

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

the syntax of natural language using typed feature structures, which are directed acyclic graphs. Within lexical semantics, especially as applied to computers, modelling word meaning is easier when a given word is understood in terms of related words; semantic networks are therefore important in computational linguistics.

- Physics and chemistry: It is used to study molecules in chemistry and physics. In condensed matter physics, the three-dimensional structure of complicated simulated atomic structures can be studied quantitatively by gathering statistics on graph-theoretic properties related to the topology of the atoms. In chemistry a graph makes a natural model for a molecule, where vertices represent atoms and edges bonds. This approach is especially used in computer processing of molecular structures, ranging from chemical editors to database searching. In statistical physics, graphs can represent local connections between interacting parts of a system, as well as the dynamics of a physical process on such systems. Similarly, in computational neuroscience graphs can be used to represent functional connections between brain areas that interact to give rise to various cognitive processes, where the vertices represent different areas of the brain and the edges represent the connections between those areas. Graph theory plays an important role in electrical modelling of electrical networks, here, weights are associated with resistance of the wire segments to obtain electrical properties of network structures.
- Social sciences: It is widely used in sociology as a way, for example, to measure the prestige of actors or to explore rumour spreading, notably through the use of social network analysis software. Under the umbrella of social networks are many different types of graphs. Acquaintanceship and friendship graphs describe whether people know each other. Influence graphs model whether certain people can influence the behaviour of others. Finally, collaboration graphs model whether two people work together in a particular way, such as acting in a movie together.
- Biology: It is useful in biology and conservation efforts where a vertex can represent regions where certain species exist (or inhabit) and the edges represent migration paths or movement between the regions. This information is important when looking at breeding patterns or tracking the spread of disease, parasites or how changes to the movement can affect other species.
- Mathematics: It is useful in geometry and certain parts of topology such as knot theory. Algebraic graph theory has close links with group theory.

13.1.7 Random graphs

See textbooks by Bollobas [1985], [1991], Frieze et al. [2015], among others.

13.1.7.1 Overview

The theory of random graphs combines the graph theory and the probability theory and are described by a probability distribution or by a random process generating them. They refer exclusively to the Erdos-Renyi random graph model (see Erdos et al. [1959], Gilbert [1959]). Aiming to describe networks seen in communications, the authors suggested that these systems could be modelled by connecting their nodes with randomly placed links. A new field emerged called random networks. These networks are used as building blocks in complex networks.
Random graphs are used to infer the properties of typical graphs. Different random graph models produce different probability distributions on graphs. They are obtained by starting with a set of n isolated vertices and randomly adding successive edges between them with probability p. We want to determine the stage where a particular property of the graph is likely to arise. Examples of random graph models are:

- The Edgar Gilbert model $G(n,p)$: A graph is constructed by connecting nodes randomly. Every possible edge occurs independently with probability p. The probability of obtaining any one particular random graph with n nodes and m edges is $p^m(1-p)^{N-m}$ with $N = \binom{n}{2} = \frac{n(n-1)}{2}$ [3]. The parameter p can be seen as a weighting function. As p increases from 0 to 1, the model becomes more likely to include graphs with more edges.

[3] The combination symbol $C_r^n = \frac{n!}{(n-r)!r!}$ is also called the binomial coefficient $\binom{n}{r}$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- The Erdos-Renyi model $G(n, M)$: It assigns equal probability to all graphs with exactly M edges. With $0 \leqslant M \leqslant N$, the model $G(n, M)$ has $\binom{N}{M}$ elements which all occurs with probability $\frac{1}{\binom{N}{M}}$. It can be viewed as a snapshot at a particular time M of the random graph process $\tilde{G}_n$, which is a stochastic process starting with n vertices and no edges, and at each step adds one new edge chosen uniformly from the set of missing edges. For $M \approx pN$, where N is the maximal number of edges possible, the two models $G(n, p)$ and $G(n, M)$ are almost interchangeable. Once we have a model of random graphs, every function on graphs becomes a random variable.

- The infinite random graph G: We start with an infinite set of vertices and let every possible edge occur independently with probability p. Except in the trivial cases $p = 0$ and $p = 1$, such a G almost surely has the following property: Given any $n + m$ elements $a_1, ..., a_n, b_1, ..., b_m \in V$, there is a vertex c in V that is adjacent to each of $a_1, ..., a_n$ and is not adjacent to any of $b_1, ..., b_m$. In the case where the vertex set is countable, there is only a single graph with this property called the Rado graph. Thus, any countably infinite random graph is almost surely the Rado graph. Hence, it is simply called the random graph. However, in the case of uncountable graphs, there are many graphs satisfying this property.

- The random dot-product model: It associates with each vertex a real vector. The probability of an edge uv between any vertices u and v is some function of the dot product $u \cdot v$ of their respective vectors.

- The network probability matrix: It models random graphs through edge probabilities representing the probability $p_{i,j}$ that a given edge $e_{i,j}$ exists for a specified time period. It is extensible to directed and undirected, weighted and unweighted, and static or dynamic graphs structure.

- Random regular graphs: They are special cases with properties differing from the random graphs.

- A random tree is a tree (see Section (4.1.2)) that is formed by a stochastic process. In a large range of random graphs of order n and size $M(n)$ the distribution of the number of tree components of order k is asymptotically Poisson. Types of random trees include uniform spanning tree, random minimal spanning tree, random binary tree, treap, rapidly exploring random tree, Brownian tree, and random forest.

- An extension of random graphs are conditional random graphs. We consider a given random graph model defined on the probability space $(\Omega, \mathcal{F}, P)$ and let $\mathcal{P}(G) : \Omega \to \mathbb{R}^m$ be a real valued function assigning to each graph in Ω a vector of m properties. For a fixed $p \in \mathbb{R}^m$, conditional random graphs are models in which the probability measure P assigns zero probability to all graphs such that $\mathcal{P}(G) \neq p$.

13.1.7.2 Description

13.1.7.2.1 The Erdos-Renyi model As discussed above, the Erdos-Renyi model and the Edgar Gilbert model are two closely related models generating random graphs. The authors defined a random graph as n nodes connected by m edges chosen randomly from the $N = \binom{n}{2} = \frac{n(n-1)}{2}$ possible edges. In total, there are $C^m_{\frac{n(n-1)}{2}}$ such graphs forming a probability space in which every realisation is equiprobable. Alternatively, we can define a random graph with the binomial model. Starting with n nodes, every pair of nodes is connected with probability p. Thus, the total number of edges is a random variable with the expectation value $E[m] = p\frac{n(n-1)}{2} = pN$. If G_0 is a graph with nodes $v_1, ..., v_n$ and m edges, the probability of obtaining it this way is $P(G_0) = p^m(1-p)^{\frac{n(n-1)}{2}-m}$.
These graphs are used in the probabilistic method to prove the existence of graphs satisfying various properties. Their behaviour is generally studied in the limit when n tends to infinity. For instance, the expected number of edges in $G(n, p)$ is Np, with $N = \binom{n}{2}$, and by the law of large numbers any graph in $G(n, p)$ will almost surely have approximately this many edges. Thus, if $pn^2 \to \infty$, then $G(n, p)$ should behave similarly to $G(n, M)$ with $M = Np$ as n increases. This is the case for many graphs.
The distribution of the degree of any particular vertex is binomial:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$P(deg(v) = k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

where n is the total number of vertices in the graph. Since

$$P(deg(v) = k) \to \frac{(np)^k e^{-np}}{k!} \text{ as } n \to \infty \text{ and } np = cst$$

in the limit, this distribution is Poisson (see Appendix (24.4.3)). Hence, random networks are also called exponential. The authors described the behaviour of $G(n,p)$ for various values of p as follows:

- if $np < 1$, then a graph in $G(n,p)$ will almost surely have no connected components of size larger than $O(\log n)$.
- if $np = 1$, then a graph in $G(n,p)$ will almost surely have a largest component whose size is of order $n^{\frac{2}{3}}$.
- if $np \to c > 1$, where c is a constant, then a graph in $G(n,p)$ will almost surely have a unique giant component containing a positive fraction of the vertices.
- if $p < \frac{(1-\epsilon)\log n}{n}$, then a graph in $G(n,p)$ will almost surely contain isolated vertices, and thus be disconnected.
- if $p > \frac{(1+\epsilon)\log n}{n}$, then a graph in $G(n,p)$ will almost surely be disconnected.

Thus, $\frac{\log n}{n}$ is a sharp threshold for the connectedness of $G(n,p)$. Further, there is a $k(n)$ (approximately equal to $2\log_2(n)$) such that the largest clique in $G(n, 0.5)$ has almost surely either $k(n)$ or $k(n)+1$.
Following the introduction to percolation above, the Erdos-Renyi process is an unweighted link percolation on the complete graph. The percolation theory is rooted in physics which refers to the study of complete graph as a mean field theory. Thus, the Erdos-Renyi process is the mean field case of percolation. Note, the relative size of the giant component P_∞ is given by

$$P_\infty = p\big(1 - e^{-<k>P_\infty}\big)$$

13.1.7.3 Some properties

The construction of a random graph is called an evolution: starting with a set of n isolated nodes, the graph develops by the successive addition of random edges. The graphs obtained at different stages of this process correspond to larger and larger connection probability p, until we get the fully connected graph. The main goal of random graph theory is to determine the probability p for which a particular property Q of a graph will most likely arise. Erdos and Renyi discovered that many important properties of random graphs appear quite suddenly. At a given probability p, either almost every graph has some property Q, or almost no graph has it. For many such properties there is a critical probability $p_c(n)$:

- if $p(n)$ grows more slowly than $p_c(n)$ as $n \to \infty$, then almost every graph with probability $p(n)$ fails to have Q.
- if $p(n)$ grows faster than $p_c(n)$ as $n \to \infty$, then almost every graph with probability $p(n)$ has the property Q.

Thus, the probability that a graph $G(n,p)$ has property Q satisfies

$$\lim_{n\to\infty} P_{n,p}(Q) = \begin{cases} 0 \text{ if } \frac{p(n)}{p_c(n)} \to 0 \\ 1 \text{ if } \frac{p(n)}{p_c(n)} \to \infty \end{cases}$$

Note, $p_c(n)$ is the critical probability in percolation and the above equation reduces to finding out whether p is smaller or larger than p_c. In finite-dimensional systems, the limit $p_c = p_c(n \to \infty)$ is obtained by finite size scaling. However,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

networks are infinite dimensional and the occupation probability is a function of the system size. Thus, we have to define a threshold function that depends on the system size. However, the average degree of a graph

$$< k >= \frac{2m}{n} = p(n-1) \approx pn$$

does have a critical value that is independent of the system size.

In order to determine the properties of random graphs we need to concentrate on their asymptotic behaviour. That is, the converging values of the probabilities as n tends to infinity. For instance, percolation theory characterises the connectedness of infinite random graphs. It is related to the robustness of the graph (or network). To do so, we consider a random graph with $n >> 1$ nodes and average degree $< k >$. Then, we remove randomly a fraction $(1-p)$ of nodes and only leave a fraction p from the network. There exists a critical percolation threshold $p_c = \frac{1}{<k>}$ below which the network becomes fragmented, while above p_c a giant connected component exists. Localised percolation consists in removing a node, then its neighbours, then the next nearest neighbours, etc. until a fraction $(1-p)$ of nodes from the network is removed.

We assume that the connection probability $p(n)$ scales as n^z, where z is a parameter taking values in $[-\infty, 0]$. When z passes $-\frac{3}{2}$, trees of order 3 suddenly appear, and as z approaches -1, the graph contains trees of larger and larger order. As long as $z < -1$, the average degree of the graph is $< k >= pn \to 0$ as $n \to \infty$, and the graph is a union of disjoint trees and cycles are absent. When z passes through -1, then $< k >= cst$, and the asymptotic probability of cycles of all orders jumps from 0 to 1. Cycles of order 3 can also be viewed as complete subgraphs of order 3. As z continues to increase, complete subgraphs of larger and larger order emerge. As z approaches 0, the graph contains complete subgraphs of all finite order.
There is also an abrupt change in the cluster structure of a random graph as $< k >$ approaches 1. If $0 < \ < k > 1$, almost surely all clusters are either trees or clusters containing exactly one cycle. The mean number of clusters is of order $n - m$. The largest cluster is a tree and its size is proportional to $\log n$. When $< k >$ passes the threshold $< k >_c= 1$, the structure of the graph changes abruptly. For $< k >_c= 1$, the largest cluster has approximately $n^{\frac{2}{3}}$ nodes with a complex structure. For $< k > \ > 1$ the greatest cluster has $\big(1 - f(< k >)\big)n$ nodes, where $f(x)$ is a function decreasing exponentially from $f(1) = 1$ to 0 for $x \to \infty$. Thus, a finite fraction $S = 1 - f(< k >)$ of the nodes belongs to the largest cluster. Except for this giant cluster, all the other ones are rather small. Hence, at $p_c \approx \frac{1}{n}$, the topology of the random graph abruptly changes from small clusters to a system dominated by a single giant cluster.

The existence of a property on a random graph can often imply the existence of that property on almost all graphs. For instance, in random regular graphs, $G(n, r-reg)$ are the set of r-regular graphs with $r = r(n)$ such that n and m are the natural numbers, $3 \leqslant r < n$, and $rn = 2m$ is even. Also, the degree sequence of a graph G in G^n depends only on the number of edges in the sets

$$V_n^{(2)} = \{ij : \ 1 \leqslant j \leqslant n, i \neq j\} \subset V^{(2)}, i = 1, ..., n$$

If the edges M in a random graph G_M is large enough to ensure that almost every G_M has minimum degree at least 1, then almost every G_M is connected and, if n is even, almost every G_M has a perfect matching. In particular, the moment the last isolated vertex vanishes in almost every random graph, the graph becomes connected. Almost every graph process on an even number of vertices with the edge raising the minimum degree 1 or a random graph with slightly more than $\frac{n}{4}\log n$ edges and with probability close to 1 ensures that the graph has a complete matching, with exception of at most one vertex. For some constant c, almost every labelled graph with n vertices and at least $cn \log n$ edges is Hamiltonian. With probability tending to one, the particular edge that increases the minimum degree to 2 makes the graph Hamiltonian. Further, properties of random graph may change or remain invariant under graph transformations.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

13.1.7.4 Some measures of topology

13.1.7.4.1 Degree distribution In a random graph with connection probability p, the degree k_i of a node i follows a binomial distribution with parameters $n-$ and p:

$$P(k_i = k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

where the probability of k edges is p^k and the probability of the absence of additional edges is $(1-p)^{n-1-k}$. If i and j are different nodes, $P(k_i = k)$ and $P(k_j = k)$ are nearly independent random variables.
In order to find the degree distribution of the graph we need to study the number of nodes with degree k, denoted X_k. We want to estimate $P(X_k = r)$. From the above equation, the expectation value of X_k is

$$E[X_k] = nP(k_i = k) = \lambda_k$$

where

$$\lambda_k = n\binom{n-1}{k} p^k (1-p)^{n-1-k}$$

The distribution of the X_k values, $P(X_k = r)$, approaches the Poisson distribution (see Appendix (24.4.3))

$$P(X_k = r) = e^{-\lambda_k} \frac{(\lambda_k)^r}{r!}$$

Thus, the number of nodes with degree k follows a Poisson distribution with mean value λ_k. If the nodes are independent, then X_k is close to $nP(k_i = k)$. Thus, the degree distribution of a random graph is approximately a binomial distribution

$$P(k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

which, for large n, tends to the Poisson distribution

$$P(k) \approx e^{-pn} \frac{(pn)^k}{k!} = e^{-<k>} \frac{<k>^k}{k!}$$

For a large range of p values, both the maximum and minimum degree are determined and finite. For instance

- if $p(n) \sim n^{-1-\frac{1}{k}}$, almost no graph has nodes with degree higher than k.
- if $p = \frac{\log n + k \log(\log n) + c}{n}$, almost every random graph has a minimum degree of at least k.

For p large enough, if $\frac{pn}{\log n} \to \infty$, the maximum degree of almost all random graphs has the same order of magnitude than the average degree. Thus, even though the position of the edges is random, a typical random graph is homogeneous, with most nodes having the same number of edges.

13.1.7.4.2 Diameter A random graph is likely to be spreading, that is, with large probability, the number of nodes at a distance l from a given node is not much smaller than $<k>^l$, for the degree k. Equating $<k>^l$ with n, the diameter is proportional to $\frac{\log n}{\log(<k>)}$, meaning that it only depends logarithmically on the number of nodes. Thus, random graphs have small diameters for p not too small. In fact, for most values of p, almost all graphs with the same n and p have the same diameter. That is, the range of values for the diameters of these graphs is concentrated around

$$d = \frac{\log n}{\log(pn)} = \frac{\log n}{\log(<k>)} \tag{13.1.1}$$

Some important results are:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- if $< k >= pn < 1$, a typical graph is composed of isolated trees and its diameter equals that of a tree.
- if $< k >> 1$, a giant cluster appears. The diameter of the graph equals that of the giant cluster if $< k >\geqslant 3.5$, and is proportional to $\frac{\log n}{\log(<k>)}$.
- if $< k >\geqslant \log n$, almost every graph is totally connected. The diameter of the graphs having the same n and $< k >$ are concentrated on a few values around $\frac{\log n}{\log(<k>)}$.

Another way of characterising the spread of a random graph is to compute the average distance between any pair of nodes, or the average path length ℓ_{rand}. It is expected that the latter scales with the number of nodes in the same way way as the diameter in Equation (13.1.1). That is,

$$\ell_{rand} \sim \frac{\log n}{\log(< k >)} \tag{13.1.2}$$

Note, given the above equation, plotting $\ell_{rand} \cdot \log(< k >)$ as a function of $\log n$ for random graphs of different sizes gives a straight line of slope (see Newman [2001]). We denote ℓ_{real} the average path length of real networks and plot $\ell_{real} \cdot \log(< k >)$ against the network size. One can show that, given the same size, ℓ_{real} and ℓ_{real} are close to each other.

13.1.7.4.3 Clustering coefficient If we consider a node in a random graph and its nearest neighbours, the probability that two of these neighbours are connected is equal to the probability that two randomly selected nodes are connected. As a result, the clustering coefficient of a random graph is

$$C_{rand} = p = \frac{< k >}{n}$$

Thus, if we plot the ratio $\frac{C_{rand}}{<k>}$ as a function of n for random graphs of different sizes on a log-log plot, they will align along a straight line of slope -1. Doing the same thing with real networks, one can see that real networks do not follow the prediction of random graphs. Rather than decreasing as n^{-1}, the fraction $\frac{C_{real}}{<k>}$ appears to be independent from n. This property is specific to large ordered lattices, whose clustering coefficient only depends on the coordination number of the lattice and not their size.

13.1.7.4.4 Graph spectra Any graph G with n nodes can be represented by its adjacent matrix $A(G)$ with $n \times n$ elements A_{ij}, with value $A_{ij} = A_{ji} = 1$ if node i and j are connected, and 0 otherwise. The spectrum of a graph is the set of eigenvalues of its adjacency matrix $A(G)$. A graph with n nodes has n eigenvalues λ_j, and its spectral density is given by

$$\rho(\lambda) = \frac{1}{n}\sum_{j=1}^{n}\delta(\lambda - \lambda_j)$$

where $\lambda(\cdot)$ is the Dirac delta function. It approaches a continuous function if $n \to \infty$. The spectral density can directly be related to the graph's topological features since its kth moment can be written as

$$\frac{1}{n}\sum_{j=1}^{n}(\lambda_j)^k = \frac{1}{n}\sum_{i_1,i_2,\ldots,i_k} A_{i_1 i_2}A_{i_2 i_3}\cdots A_{i_k i_1}$$

which is the number of paths returning to the same node in the tree.
Given a random graph $G(n,p)$ satisfying $p(n) = cn^{-z}$. For $z < 1$, there is an infinite cluster in the graph, and as $n \to \infty$, any nodes belongs to the infinite cluster almost surely. In that case, the spectral density converges to a semicircular distribution

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\rho(\lambda) = \begin{cases} \frac{\sqrt{4np(1-p)-\lambda^2}}{2\pi np(1-p)} \text{ if } |\lambda| < 2\sqrt{np(1-p)} \\ 0 \text{ otherwise} \end{cases}$$

This is the Wigner's law (or semicircle law) (see Wigner [1958]). The largest eigenvalue λ_1 is isolated from the bulk of the spectrum, and it increases with the network size as np. When $z > 1$, the spectral density deviates from the semicircle law. Further, the odd moments of the density are equal to zero, so that a path can only come back to the original node is if it returns following exactly the same nodes, which is a feature of tree structure.

13.2 Graphical models

See books by Jordan [1999], Koller et al. [2009], Russell et al. [2009], Barber [2012], among other.

13.2.1 Description

A graphical model, or probabilistic graphical model (PGM), is a probabilistic model for which a graph (see Section (13.1.1.2)) expresses the conditional dependence structure between random variables. They are commonly used in probability theory, statistics, particularly Bayesian statistics, and machine learning.
In general, probabilistic graphical models use a graph-based representation as the foundation for encoding a distribution over a multi-dimensional space and a graph that is a compact or factorised representation of a set of independences that hold in the specific distribution. Two branches of graphical representations of distributions are commonly used. These graphical models are:

- Bayesian networks: It is a probabilistic graphical model representing a set of variables and their conditional dependencies via a directed acyclic graph (DAG).
- Markov random fields (MRF): It a set of random variables having a Markov property described by an undirected graph.

Both families encompass the properties of

1. factorisation, and
2. independences,

but they differ in the set of independences they can encode and the factorisation of the distribution that they induce.
Some other types of graphical models are:

- Factor graph: It is an undirected bipartite graph connecting variables and factors. Each factor represents a function over the variables it is connected to. This is a helpful representation for understanding and implementing belief propagation.
- Clique tree or junction tree: It is a tree of cliques, used in the junction tree algorithm.
- Chain graph: It is a graph which may have both directed and undirected edges, but without any directed cycles (if we start at any vertex and move along the graph respecting the directions of any arrows, we cannot return to the vertex we started from if we have passed an arrow). Both directed acyclic graphs and undirected graphs are special cases of chain graphs, which can therefore provide a way of unifying and generalising Bayesian and Markov networks.
- Ancestral graph: It is a further extension, having directed, bidirected and undirected edges.
- Random field techniques

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

  - A Markov random field, also known as a Markov network, is a model over an undirected graph. A graphical model with many repeated subunits can be represented with plate notation.
  - A conditional random field is a discriminative model specified over an undirected graph.

- Restricted Boltzmann machine: It is a bipartite generative model specified over an undirected graph.

Graphical models provide algorithms for discovering and analysing structure in complex distributions. They can describe the distributions succinctly and extract the unstructured information. Applications of graphical models include causal inference, information extraction, speech recognition, computer vision, decoding of low-density parity-check codes, modelling of gene regulatory networks, gene finding and diagnosis of diseases, and graphical models for protein structure.

13.2.2 Bayesian networks

See textbook by Russell et al. [2009], among others.

13.2.2.1 Description

A Bayesian network, belief network, decision network, or probabilistic directed acyclic graphical model is a probabilistic graphical model that represents a set of variables and their conditional dependencies via a directed acyclic graph (DAG) (see Section (13.1.3)). Bayesian networks are ideal for taking an event that occurred and predicting the likelihood that any one of several possible known causes was the contributing factor. For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.
Bayesian networks that model sequences of variables (such as speech signals or protein sequences) are called dynamic Bayesian networks. Generalisations of Bayesian networks that can represent and solve decision problems under uncertainty are called influence diagrams.

Formally, Bayesian networks are DAGs whose nodes represent variables in the Bayesian sense: they may be observable quantities, latent variables, unknown parameters or hypotheses (see Pearl [1985]).
Edges represent conditional dependencies; nodes that are not connected (no path connects one node to another) represent variables that are conditionally independent of each other. Any two nodes are conditionally independent given the values of their parents. In general, any two sets of nodes are conditionally independent, given a third set, if a criterion called d-separation holds in the graph. Local independences and global independences are equivalent in Bayesian networks.
Each node is associated with a probability function that takes, as input, a particular set of values for the node's parent variables, and gives (as output) the probability (or probability distribution, if applicable) of the variable represented by the node.
For example, if m parent nodes represent m Boolean variables then the probability function could be represented by a table of 2^m entries, one entry for each of the 2^m possible parent combinations. Similar ideas may be applied to undirected, and possibly cyclic, graphs such as Markov networks.
This type of graphical model is known as a directed graphical model, Bayesian network, or belief network. Classic machine learning models like hidden Markov models, neural networks and newer models such as variable-order Markov models can be considered special cases of Bayesian networks.

13.2.2.2 Some concepts

13.2.2.2.1 Factorisation In the case where the network structure of the model is a directed acyclic graph (DAG) (see Section (13.1.3)), the model represents a factorisation of the joint probability of all random variables (see Appendix (24.1.2)).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

More precisely, we let $G = (V, E)$ be a DAG and let $X = (X_v)$, $v \in V$ be a set of random variables indexed by v. Then, X is a Bayesian network with respect to G if its joint probability density function (pdf) can be written as a product of the individual density functions, conditional on their parent variables. That is, the joint pdf $f_X(x) = p(x)$ is given by

$$p(x) = \prod_{v \in V} p(x_v | \mathcal{K}_v) \tag{13.2.3}$$

where $\mathcal{K}_v$ is the set of parents of v. The vertices are pointing directly to v via a single edge. That is, the joint distribution factors into a product of conditional distributions. For example, the graphical model in the Figure (13.6) (which is actually not a directed acyclic graph, but an ancestral graph) consists of the random variables A, B, C, D with a joint probability density that factors as

$$P(A, B, C, D) = P(A) \cdot P(B) \cdot P(C, D|A, B)$$

Figure 13.6: Graphical model where each arrow indicates a dependency.

For any set of random variables $X_1, ..., X_n$, then the probability of any member of a joint probability distribution can be calculated from conditional probabilities using the chain rule (see Equation (24.1.4)) (given a topological ordering of X, see Section (13.1.4)) as follows:

$$P(X_1 = x_1, ..., X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i | X_{i+1} = x_{i+1}, ..., X_n = x_n)$$

Using the definition of the Bayesian network in Equation (13.2.3), the joint distribution can be rewritten as

$$P(X_1 = x_1, ..., X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i | X_j = x_j \text{ for each } X_j \text{ which is a parent of } X_i)$$

The difference between the two expressions is the conditional independence of the variables from any of their non-descendants, given the values of their parent variables.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

13.2.2.2.2 The local Markov property We can say that X is a Bayesian network with respect to G if it satisfies the local Markov property. That is, each variable is conditionally independent of its non-descendants given its parent variables:

$$X_v \perp X_{V \backslash de(v)} \mid X_{\mathcal{K}_v} \text{ for all } v \in V \tag{13.2.4}$$

where $de(v)$ is the set of descendants and $V \backslash de(v)$ is the set of non-descendants of v. We can express this relationship in terms of conditional probability as follows:

$$P\big(X_v = x_v \mid X_i = x_i \text{ for each } X_i \text{ which is not descendant of } X_v\big) = P\big(X_v = x_v \mid X_j = x_j \text{ for each } X_j \text{ which is a parent of } X_v\big)$$

The set of parents is a subset of the set of non-descendants since the graph is acyclic.

13.2.2.2.3 The Markov blanket The Markov blanket of a node is the set of nodes consisting of its parents, its children, and any other parents of its children. It renders the node independent of the rest of the network. Thus, the joint distribution of the variables in the Markov blanket of a node is sufficient knowledge for calculating the distribution of the node. X is a Bayesian network with respect to G if every node is conditionally independent of all other nodes in the network, given its Markov blanket.
This definition can be made more general by defining the d-separation of two nodes, where d stands for directional. Let P be a trail from node u to v, where a trail is a loop-free, undirected path between two nodes. Then, P is said to be d-separated by a set of nodes Z if any of the following conditions holds:

- P contains a directed chain, $u \cdots \leftarrow m \leftarrow \cdots v$ or $u \cdots \rightarrow m \rightarrow \cdots v$, such that the middle node m is in Z.
- P contains a fork, $u \cdots \leftarrow m \rightarrow \cdots v$, such that the middle node m is in Z.
- P contains an inverted fork (or collider), $u \cdots \rightarrow m \leftarrow \cdots v$, such that the middle node m is not in Z and no descendant of m is in Z.

The nodes u and v are d-separated by Z if all trails between them are d-separated. If u and v are not d-separated, they are d-connected. Thus, X is a Bayesian network with respect to G if, for any two nodes u and v, we get

$$X_u \perp X_v \mid X_Z$$

where Z is a set which d-separates u and v. Note, the Markov blanket is the minimal set of nodes which d-separates node v from all other nodes.

13.2.2.2.4 Causal networks Bayesian networks may not always represent causal relationships. For instance, a directed edge from u to v does not require that X_v be causally dependent on X_u. It can be proved by the fact that Bayesian networks on the graphs

$$a \rightarrow b \rightarrow c \text{ and } a \leftarrow b \leftarrow c$$

are equivalent, that is, they impose exactly the same conditional independence requirements.
A causal network is a Bayesian network with the requirement that the relationships be causal. The additional semantics of causal networks specify that if a node X is actively caused to be in a given state x ($X = x$), then the probability density function changes to that of the network obtained by cutting the links from the parents of X to X, and setting X to the caused value x. We can use these semantics to predict the impact of external interventions from data obtained prior to intervention.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

13.2.2.2.5 Implementation When implementing a Bayesian network, we often start with creating a DAG G such that X satisfies the local Markov property with respect to G. Sometimes this is a causal DAG. The conditional probability distributions of each variable given its parents in G are assessed. In many cases (specially with discrete variables), if the joint distribution of X is the product of these conditional distributions, then X is a Bayesian network with respect to G.

13.2.2.2.6 Approximation algorithms Cooper [1990] proved that exact inference in Bayesian networks was NP-hard. This result prompted research on approximation algorithms with the aim of developing a tractable approximation to probabilistic inference. Dagum et al [1993] proved two surprising results on the complexity of approximation of probabilistic inference in Bayesian networks:

1. they proved that no tractable deterministic algorithm can approximate probabilistic inference to within an absolute error $\epsilon < \frac{1}{2}$.
2. they proved that no tractable randomized algorithm can approximate probabilistic inference to within an absolute error $\epsilon < \frac{1}{2}$ with confidence probability greater than $\frac{1}{2}$.

In practical terms, these complexity results suggested that while Bayesian networks were rich representations for AI and machine learning applications, their use in large real-world applications would need to be tempered by either topological structural constraints, such as naÃ¯ve Bayes networks, or by restrictions on the conditional probabilities. The bounded variance algorithm (see Dagum et al. [1997]) was the first provable fast approximation algorithm to efficiently approximate probabilistic inference in Bayesian networks with guarantees on the error approximation. This powerful algorithm required the minor restriction on the conditional probabilities of the Bayesian network to be bounded away from zero and one by $\frac{1}{p(n)}$ where $p(n)$ is any polynomial on the number of nodes in the network n.

13.2.2.3 Inference tasks

Bayesian networks (BNs) perform three main inference tasks:

1. Inferring unobserved variables: BN can be used to answer probabilistic queries about its variables and their relationships. The network can be used to update knowledge of the state of a subset of variables when other variables (the evidence variables) are observed. This process of computing the posterior distribution of variables given evidence is called probabilistic inference. The posterior gives a universal sufficient statistic for detection applications, when choosing values for the variable subset that minimise some expected loss function, for instance the probability of decision error. Thus, a BN can be considered a mechanism for automatically applying Bayes' theorem to complex problems.
2. Parameter learning: In order to fully specify the BN and thus fully represent the joint probability distribution, it is necessary to specify for each node X the probability distribution for X conditional upon X's parents. The distribution of X conditional upon its parents may have any form. It is common to work with discrete or Gaussian distributions since that simplifies calculations. Sometimes only constraints on a distribution are known; one can then use the principle of maximum entropy to determine a single distribution, the one with the greatest entropy given the constraints.
   In general, these conditional distributions include parameters that are unknown and must be estimated from data, for example via the maximum likelihood approach. Direct maximisation of the likelihood (or of the posterior probability) is often complex given unobserved variables. A classical approach to this problem is the expectation-maximisation algorithm, which alternates computing expected values of the unobserved variables conditional on observed data, with maximising the complete likelihood (or posterior) assuming that previously computed expected values are correct. Under mild regularity conditions this process converges on maximum likelihood (or maximum posterior) values for parameters.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3. Structure learning: In the simplest case, a Bayesian network is specified by an expert and is then used to perform inference. In other applications the task of defining the network is too complex for humans. In this case the network structure and the parameters of the local distributions must be learned from data.
   Automatically learning the graph structure of a Bayesian network (BN) is a challenge pursued within machine learning. The basic idea goes back to a recovery algorithm developed by Rebane et al. [1987] and rests on the distinction between the three possible patterns allowed in a 3-node DAG.
   An alternative method of structural learning uses optimisation-based search. It requires a scoring function and a search strategy. A common scoring function is posterior probability of the structure given the training data, like the BIC or the BDeu. The time requirement of an exhaustive search returning a structure that maximises the score is superexponential in the number of variables. A local search strategy makes incremental changes aimed at improving the score of the structure.
   A particularly fast method for exact BN learning is to cast the problem as an optimisation problem, and solve it using integer programming. Acyclicity constraints are added to the integer program (IP) during solving in the form of cutting planes.

13.3 Complex networks

So far, we have considered networks, in Section (13.1), where the connection topology is assumed to be either completely regular or completely random. Further, we have seen in Section (13.1.7) that most nodes in a random network have approximately the same number of links. In the limit, they follow a Poisson distribution with a bell shape. However, several real networks lie somewhere between these two extremes. That is, some nodes have a very large number of connections to other nodes (these nodes are called hubs), while most nodes have just a handful.
A complex network is a graph network with non-trivial topological features that do not occur in simple networks, such as lattices or random graphs, but often occur in graphs modelling of real systems. Most social, biological, and technological networks display substantial non-trivial topological features, with patterns of connection between their elements that are neither purely regular nor purely random. Such features include a heavy tail in the degree distribution, a high clustering coefficient, assortativity or disassortativity among vertices, community structure, and hierarchical structure (see Section (13.1.1.4)). This is illustrated in Figure (13.7).

Figure 13.7: Complex networks.

Complex networks can represent any system with interactions in a unified way by stripping complicated details andretaining the main features of the system. The resulting networks only include nodes, representing the interacting

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

agents, and links, representing interactions. Many real networks have two fundamental properties: If the degree distribution of the network follows a power law, it is scale-free. If any two arbitrary nodes can be connected in a very small number of steps, it is small-world. As a result, two well known examples of complex networks are small-world networks (see Watts et al. [1998], Stanley et al. [2000]), characterised by short path lengths and high clustering, and scale-free networks (see Barabasi [1999], Albert et al. [2002]), characterised by power-law degree distributions. Many other aspects of network structure have been studied. For instance, hierarchical network models grow according to some patterns. Alternatively, the Yule process is a canonical generative process for power laws. Recently, Hyperbolic Geometric Graphs have been suggested as yet another way of constructing scale-free networks. Assuming that a network has an underlying hyperbolic geometry, we can use the framework of spatial networks to generate scale-free degree distributions. The heterogeneous degree distribution reflects the negative curvature and metric properties of underlying hyperbolic geometry (see Krioukov et al. [2010]).

Originally, complex networks have been studied by ignoring the details of their individual links and nodes. However, in order to understand the mechanism responsible for the emergence of scale-free networks, Barabasi et al. [2001], Albert et al. [2002] showed that one had to shift from modelling network topology to modelling the network assembly and evolution. In the latter, the emphasis is on capturing the network dynamics. The idea being that if we correctly assess the process assembling the networks at a given time, we also infer their topology. It involves determining the time at which each node joined the network and is related to stochastic calculus. Thus, one needs to model the dynamics of networks to reflect well known properties of complex dynamical systems. For instance, fractal features and self-similarity, common in real systems, are also known to influence the dynamic processes taking place in networks. Rozenfeld et al. [2008] reviewed the recent advances in self-similarity of complex networks and their relations to topological such as degree distribution and degree-degree correlations. Peng et al. [2017] proposed a class of scale-free deterministic networks, called (u, v)-flowers, with parameters u, v controlling the topological properties.

13.3.1 Generalised random graphs

Since random graphs do not capture the scale-free character of real networks, generalised random graphs developed as semi-random graphs, where the degree distribution is an input and the rest of the model is random. That is, the edges connect randomly selected nodes, with the constraint that the degree distribution is restricted to a power law.

13.3.1.1 Measures of topology

13.3.1.1.1 Clustering coefficient Complex networks exhibit a large degree of clustering. It can be formalised as follows:

Definition 13.3.1 *Clustering coefficient*
Given a graph $G = (V, E)$, the clustering coefficient of a vertex v, denoted $C(v)$ is the fraction, over all pairs of neighbour of v, of those pairs who are neighbours of each other. Formally,

$$C(v) = \frac{\left|\{(u, w) \in E : \ u, w \in \mathcal{N}(v)\}\right|}{\binom{d(v)}{2}}$$

Put another way, the clustering coefficient is the probability that, picking two friends uniformly at random, they are also friends. Note, the clustering coefficient of a graph is the average clustering coefficient of its nodes. We see that the clustering coefficient of trees and cycles is zero. On the other hand, the clustering coefficient of social networks is high and ideally constant (independent of the graph size).

13.3.1.1.2 The degree distribution The degree distribution is very important in studying both real networks, such as the Internet and social networks, and theoretical networks.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

For example, we saw in Section (13.1.7.4) that a random graph, in which each of n nodes is independently connected (or not) with probability p (or $1-p$), has a binomial distribution of degrees k:

$$P(k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

If the average degree $< k >= p(n-1)$ is held fixed and we take the limit of large n, we get a Poisson distribution with peak at $P(< k >)$. However, most real world networks are highly right-skewed, meaning that a large majority of nodes have low degree but a small number, known as 'hubs', have high degree.
Some networks, notably the Internet, the world wide web (WWW), and some social networks are found to have degree distributions that approximately follow a power law (see Appendix (24.4.5))

$$P(k) \sim k^{-\gamma}$$

where γ is a constant. Such networks are called scale-free networks and have attracted particular attention for their structural and dynamical properties (see Albert et al. [1999], Kumar et al. [1999] Pachon et al. [2018]). The nodes of the world wide web (WWW) network are the documents (web pages) and the edges are the hyperlinks (URLs) pointing from one document to another. The latter being directed, the network is characterised by two degree distribution, the distribution of outgoing edges $P_{out}(k)$ with γ_{out} and that of incoming edges $P_{in}(k)$ with γ_{in}.

13.3.1.2 Social networks

Similarly to brains, societies are networks of people connected by friendships, familial and professional relationships. We are interested in understanding their structure and properties. In order to find models that can best describe social networks we need to understand the topological properties explaining phenomena such as the small world hypothesis. One explanation to social networks is the small-world hypothesis by Milgram, which states that two arbitrary people are connected by only six degree of separation, that is, the diameter of the corresponding graph of social connection is not much larger than six (short path). Thus, in most networks there is a relatively short path between any two nodes (small diameter). This property seems to characterise most complex networks.
Another common property of social networks is the formation of cliques, representing circles of friends or acquaintances in which every member knows every other member. This is quantified by the clustering coefficient. The latter seems to be independent of the network size, which is a property of ordered lattices. Some existing models having these properties are:

- The clique: It is a graph in which every node is connected to every other node in the graph. In that case the degree of every node is very large.
- Trees with constant degree: However, our friends are likely to be friends themselves.

13.3.1.3 Building blocks

We are now going to present two types of networks that can be used as building blocks for social networks.

13.3.1.3.1 Ring lattices A ring lattice is a graph obtained by taking a cycle and connecting each vertex (node) to its neighbours two hops away, producing a 4-regular graph. Figure (13.8) illustrates a 4-regular lattice on twelve vertices. It can be generalised to other even constants greater than four. One can show that the clustering coefficient of a k-regular ring lattice is

$$C = \frac{3}{4}\frac{(k-2)}{(k-1)}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

which converges to $\frac{3}{4}$ in the limit of large k. However, the diameter of a k-regular lattice is $\Theta(\frac{n}{k})$ [4]. Hence, the ring lattice has a high clustering coefficient, but it also has a large diameter.

Figure 13.8: A 4-regular lattice on twelve vertices

13.3.1.3.2 Random graphs Random graphs are presented in Section (13.1.7). Since the edges are randomly distributed, the clustering coefficient is $C = p$ (this is not large enough). Following the above discussion, we would like to formalise a graph property that will guarantee small diameter.

Definition 13.3.2 *The expansion of a graph $G = (V, E)$ is the minimum, over all cuts we can make (dividing the graph in two pieces), of the number of edges crossing the cut divided by the number of vertices in the smaller half of the cut. It is given by*

$$\alpha = \min_{S \subseteq V, 1 \leqslant |S| \leqslant \frac{n}{2}} \frac{|e(S)|}{|S|}$$

where $e(S)$ is the set of edges leaving the set of nodes S.

Theorem 13.3.1 *Suppose the graph G is d-regular, for some $d \geqslant 3$ and has expansion α. Then, the diameter of G is $O(\frac{d}{\alpha} \log n)$.*

This theorem implies that for $d \geqslant 3$, a d-regular graph with constant expansion α produces the small world property. It has logarithmic diameter. We can use randomness to create d-regular graphs with good expansion. Actually, for $d \geqslant 3$ then d-random graphs have expansion that is only a function of d, and not the number of vertices. However, even though d-regular random graphs have small diameter, they have low clustering coefficients (about $\frac{d}{n}$).

[4] There are constants $c_1 < c_2$ so that the diameter of k-regular ring lattices on n is between $c_1 \frac{n}{k}$ and $c_2 \frac{n}{k}$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

13.3.1.3.3 A small world network In order to build a network with small diameter and high clustering coefficient we can combine the above two models for some constant $k \geqslant 3$. We can consider a one-parameter model interpolating between an ordered fine dimensional lattice and a random graph. For instance, we can take a K-regular random graph and add the ring lattice edges to the nodes of the random graph. We obtain a graph with a constant degree having both short distances and high clustering coefficient. This is a variant of the model proposed by Watts et al. [1998]. They took the ring lattice and rewired every edge to a random node in the graph with some probability p. The algorithm is as follows:

1. Start with order: start with a ring lattice with n nodes, where every node is connected to its first k neighbours ($\frac{k}{2}$ on either side). In order to have a sparse but connected network at all times, we consider $n >> k >> \log n >> 1$.

2. Randomise: randomly rewire each edge of the lattice with probability p such that self-connections and duplicate edges are excluded. It introduces $\frac{pnk}{2}$ long-range edges which connect nodes that otherwise would be part of different neighbourhoods.

By varying p, we control the transition between order $p = 0$ and randomness $p = 1$.

13.3.2 Small-world networks

13.3.2.1 Overview

As discussed above, Watts et al. [1998] proposed the small-world networks which are regular networks rewired to introduce increasing amounts of disorder through a single parameter smoothly interpolating between a random graph and a lattice. They demonstrated that with the addition of only a small number of long-range links, a regular graph, in which the diameter is proportional to the size of the network, can be transformed into a small world in which the average number of edges between any two vertices is very small (mathematically, it should grow as the logarithm of the size of the network), while the clustering coefficient stays large. That is, these systems can be highly clustered, like regular lattices, yet have small characteristic path lengths, like random graphs. Models of dynamical systems with small-world coupling display enhanced signal-propagation speed, computational power, and synchronisability. In particular, infectious diseases spread more easily in small-world networks than in regular lattices. A wide variety of abstract graphs exhibit the small-world property, such as random graphs and scale-free networks. Further, real world networks such as the world wide web (WWW) and the metabolic network also exhibit this property.

13.3.2.2 Description

Watts et al. [1998] presented their interpolation technique between regular and random networks by starting from a ring lattice with n vertices and k edges per vertex. They chose a vertex and the edge connecting it to its nearest neighbour in a clockwise sense. They reconnected this edge to a vertex chosen randomly with probability p based on a uniform law. This process is repeated until the ring is rewired. Since there are $\frac{nk}{2}$ edges in the entire graph, the process stops after $\frac{k}{2}$. When $p = 0$ we get the original ring lattice and when $p = 1$ we get a random graph. We get a small world network for intermediary values $0 < p < 1$. This approach is illustrated in Figure (13.9).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 13.9: $n = 20$ and $k = 4$

The structural properties of these graphs are summarised by two measures (see Section (13.1.7.4)) expressed as a function of the rewiring probability p:

1. The characteristic path length $L(p)$: It measures the typical separation between two vertices in the graph. It is a global measure.
2. The clustering coefficient $C(p)$: It measures the cliquishness of a typical neighbourhood. It is a local measure.

The authors were interested in a network having many vertices with sparse connections, but not a disconnected one. Thus, we must have $n >> k >> \log n >> 1$. They found the two limiting cases:

- $L \sim \frac{n}{2k} >> 1$ and $C \sim \frac{3}{4}$ as $p \to 0$. At $p = 0$, the lattice is a highly clustered, large world where L grows linearly with the system size n.
- $L \approx L_{random} \sim \frac{\log n}{\log k}$ and $C \approx C_{random} \sim \frac{k}{n}$ as $p \to 1$. At $p = 1$, the network is a poorly clustered, small world where L grows logarithmically with n.

There is a broad range of p where $L(p)$ is almost as small as L_{random}, but $C(p) >> C_{random}$. Thus, the small-world networks result from the immediate drop in $L(p)$ caused by the introduction of a few long-range edges. These short cuts connect vertices that are much farther apart than L_{random}. For small p, each short cut has a highly nonlinear effect on L, contracting the distance between pairs of vertices, and their surrounding. On the other hand, removing an edge from a cluster has at most a linear effect on C. Thus, $C(p)$ hardly changes for small p, even though $L(p)$ drops rapidly. At the local level, the transition to a small world is hardly detectable.
Newman et al. [1999a] [1999b] proposed a variant of the WS-model where edges are added between randomly chosen pairs of sites, but no edges are removed from the regular lattice. Contrary to the WS-model, this model does not lead to the formation of isolated clusters.

13.3.3 Scale-free networks

13.3.3.1 Description

13.3.3.1.1 Overview In 1965 Price studied the network of citations between scientific papers, and discovered that both the in-and-out degree of a citation network (see Section (13.1.1.4)) had power-law distributions, making this the first published example of a scale-free network. Further, Price [1976] developed a mathematical theory of the growth of citation networks based on what is now called a preferential attachment process.
Interest in scale-free networks began in the late 1990s with the reporting of discoveries of power-law degree distributions in real world networks. For instance, Barabasi [1999] and Kumar et al. [1999] mapped the topology of a

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

portion of the world wide web (WWW) and found that some nodes had many more connections (called hubs) than others and that the network had a power-law distribution of the number of links connecting to a node. It implies that the degree distribution of these networks has no characteristic scale. These networks were called scale-free. This result also applied to other networks, such as social and biological networks (see Barabasi et al. [2003]). However, statistical analysis has refuted many of these claims.
This is to be compared with networks having a single scale, where every node has roughly the same degree. In a network with a scale-free degree distribution, some vertices have a degree that is orders of magnitude larger than the average. These vertices are often called 'hubs', although this is a bit misleading as there is no inherent threshold above which a node can be viewed as a hub. If there were such a threshold, the network would not be scale-free.

As discussed in Section (13.3.2), the WS-model start with a fixed number n of vertices that are randomly connected or rewired, without modifying n. However, open systems grow by the continuous addition of new nodes over time. For instance, the WWW grows exponentially in time with the addition of new web pages. Further, random graphs and the WS-model assume independence of the degree of the nodes (new edges are placed randomly). On the other hand, real networks exhibit preferential attachment such that the probability of connection of a node depends on its degree. Thus, combining these two components, growth and preferential attachment, the BA-model (see Barabasi [1999]) generated a power-law degree distribution.

13.3.3.1.2 Some properties One need to study the growth of graph networks (adding new nodes) and define their characteristics. The most common characteristic in a scale-free network is the relative commonness of vertices with a degree that greatly exceeds the average. Nodes with a higher degree have a stronger ability to grab links added to the network. The major hubs are closely followed by smaller ones which in turns are followed by other nodes with an even smaller degree and so on. This hierarchy allows for a fault tolerant behaviour. That is, the scale-free property strongly correlates with the robustness to failure of a network. Some networks with a power-law degree distribution can be highly resistant to the random deletion of vertices. If failures occur at random and most nodes have small degree, the likelihood that a hub would be affected is almost negligible. Even if a hub was failing, the network would not loose its connectedness due to the remaining hubs. Cohen et al. [2000] used percolation theory to study these properties and proved that for a broad range of scale-free networks the critical percolation threshold was $p_c = 0$. That is, removing randomly any fraction of nodes from the network will not destroy the network. However, relying on hubs may problematic as they can be vulnerable to attacks. Some networks can be quite sensitive to targeted attacks aimed at fracturing the network quickly. For instance, when the graph is uniformly random except for the degree distribution, these critical vertices are the ones with the highest degree, and have thus been implicated in the spread of disease (natural and artificial) in social and communication networks, and in the spread of fads.
Further, scale-free networks exhibit the clustering coefficient distribution which decreases as the node degree increases. This distribution follows a power-law which implies that the low-degree nodes belong to very dense sub-graphs connected to each other through hubs. For example, given a social network where the nodes are people and the links are acquaintance relationships between people, the people tend to form communities. However, some people (celebrities, politicians) are connected to a large number of communities. They represent the hubs responsible for the small-world phenomenon.
Another characteristic is the average distance between two vertices in a network. While random graphs (ER) have an average distance of order $\log N$ between nodes, where N is the number of nodes, scale-free graphs can have a distance of $\log \log N$. Such graphs are called ultra-small-world networks.

13.3.3.2 Building networks

There are many different ways for building a network with a power-law degree distribution. As discussed in Section (11.2.4.2), generative models are models of the conditional probability of the observable X, given a target y. A generative mechanism called preferential attachment was proposed (see Price [1976], Barabasi [1999]) to explain the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

appearance of power-law distribution. The authors considered two major components explaining the emergence of the scale-free property in complex networks:

1. The growth process: Over an extended period of time, new nodes join an already existing system.
2. The preferential attachment: A new coming node prefers to connect to another node which has already a certain number of links with other nodes. Thus, there is a higher probability that more nodes will link themselves to that one which has already many links, leading this node to a hub. The latter may be either assortative or disassortative.

The preferential attachment is an example of a positive feedback cycle where initially random variations (one node initially having more links) are automatically reinforced, greatly magnifying differences. Following an approach innitiated by Price called cumulative advantage, Barabasi [1999] proposed the rich get richer model in which each new web page creates links to existing web pages with a probability distribution which is not uniform, but proportional to the current in-degree of web pages. The mechanism of preferential attachment tends to be linear, that is, a new node is twice as likely to link to an existing node that has twice as many connections as its neighbour.

We denote π the probability that new nodes will be connected to old ones. In general, for a network with degree k, we have the nonlinear form

$$\pi(k) = A + k^{\alpha}$$

where A is the initial attractiveness of the node (see Jeong et al. [2001]).
The algorithm of the Barabasi-Albert model (BA-model) (see Albert et al. [2002]) is as follows:

1. Growth: Start with an initial connected network of m_0 nodes. Then, the model adds one new node at every time step. Each new node is connected to $m \leqslant m_0$ existing nodes with a probability that is proportional to the number of links that the existing nodes already have.
2. Preferential attachment: We assume that the probability p_i that a new node is connected to node i depends on the degree k_i of node i is given by

$$p_i = \pi(k_i) = \frac{k_i}{\sum_j k_j}$$

where the sum is made over all pre-existing nodes j. Note, the functional form is linear in k_i.

After t time steps, this process results in a network with $n = t + m_0$ nodes and mt edges. Numerical simulations indicated a scale-invariant state and power law distribution with exponent $\gamma_{BA} = 3$.
The authors tested whether both, growth and preferential attachment (PA), were necessary in the development of networks. They considered two models: Model A keeps the growing character of the network with no PA, and model B starts with n nodes and no edges, eliminating the growth process. After some investigations, they concluded that growth and PA were simultaneously needed to reproduce the stationary power-law distribution observed in real networks.

13.3.3.3 Properties of the BA-model

The authors found that the average path length of the BA-model increased approximately logarithmically with size of the network n, with a best fit having the form

$$\ell = A \log(n - B) + C$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

for some constants A, B, C. Bollobas et al. [2004] found that the actual form has a double logarithmic correction to the logarithmic dependence given by

$$\ell \sim \frac{\log n}{\log \log n}$$

In the $\log \log$ scale, the power law function is a straight line [5].
Considering a network with average degree $< k >= 4$ and different sizes, the authors compared the clustering coefficient of the BA-model with that of a random graph and found that the coefficient of the former was about five times that of the latter, and that this factor slowly increased with number of nodes.However, the coefficient decreased with the network size n, following approximately a power law $C \sim n^{-0.75}$, which is different from that of the small-world model, where C is independent of n.
Numerical simulations indicate that the bulk of the spectral density ρ had a triangle-like shape with the top lying well above the semicircle and edges decaying as a power law. In the BA-model, the fraction of loops with more than four edges increases with n, and the growth rate of the loops increases with their size. Also, the fraction of triangles decreases as $n \to \infty$.

13.3.3.4 Evolving networks

In order to understand the mechanism responsible for the emergence of scale-free networks, we need to model the network assembly and evolution. The emphasis is on capturing the network dynamics. For instance, since the BA-model describes a time developing phenomenon, we can look for its dynamic scaling property. Compared to real networks, the BA-model has obvious limitations. For example, it predicts a lower-law degree distribution with a fixed exponent, while that measured on some networks vary between 1 to 3. Further, the degree distribution of real networks can have non power-law features such as exponential cutoffs or saturation for small k. Thus, more complex models are needed and a proper theory of evolving networks need to be proposed.

In the BA-model, the probability that new nodes will be connected to old ones, $\pi(k)$, involves two assumptions: first that it depends on k, second that the functional form is linear in k. Note, that form can be determined when we know the time at which each node joined the network. At time t record the number of old nodes in the network. Then measure the increase in the degree of the old nodes over a time interval ΔT. Plot the relative increase $\frac{\Delta k_i}{\Delta k}$ as a function of the earlier degree k_i, where Δk is the number of edges added to the network over ΔT. The obtained $\pi(k)$ follows a power law $\pi(k) \sim k^{\alpha}$. Depending on the value of α, distinct phases have been identified: sublinear case ($\alpha < 1$) and superlinear case ($\alpha > 1$). Krapivsky et al. [2000] showed that the scale-free nature of the network was destroyed for nonlinear preference attachment. Only when it is asymptotically linear is the topology of the network scale-free, that is, $\pi(k_i) \sim a_{\infty} k_i$ as $k_i \to \infty$. In that case, the rate equation leads to

$$P(k) \sim k^{-\gamma} \text{ with } \gamma = 1 + \frac{\mu}{a_{\infty}}$$

and the exponent of the degree distribution can vary in the range $[2, \infty]$.
Another feature of $\pi(k)$ in real networks is $\pi(0) \neq 0$, that is, there is a non-null probability that a new node connects to an isolated node (see Jeong et al. [2001]). Thus,

$$\pi(k) = A + k^{\alpha}$$

where A is the initial attractiveness of the node i. It describes the likelihood that an isolated node will be discovered.
In the BA-model, the number of nodes and edges increases linearly in time, so that the average degree of the network is constant. However, the growth rate on real networks can be nonlinear. In several systems, the number of edges increases faster than the number of nodes, supporting the presence of a phenomenon called accelerated growth. Dorogovtsev et al. [2000] proposed a model where at each time step a new node is added to the network, which

[5] When plotted on a double-logarithmic scale, a power law is a straight line.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

receives n incoming edges from random nodes in the system. In addition, $c_0 t^\theta$ new edges are distributed, each of them directed from a randomly selected node to a node with high incoming degree, with asymptotically linear preferential attachment $\pi(k_{in}) \propto A + k_{in}$. The scale-free nature of the degree distribution is not changed, but the degree exponent becomes

$$\gamma = 1 + \frac{1}{1+\theta}$$

Barabasi et al. [2001] proposed an undirected model where new nodes are added to the system with a constant rate, but they are connected to b nodes already in the system with preferential attachment

$$P(k_i) = b\frac{k_i}{\sum_j k_j}$$

In addition, at each time step a linearly increasing number of edges are distributed between the nodes. The probability that an edge is added between nodes i and j is

$$P_{ij} = \frac{k_i k_j}{\sum_{s,l} k_s k_l} N(t) a$$

where $N(t)$ is the number of nodes in the system, and the summation is over all nonequal values of s and l. From these two processes, the average degree of the network increases linearly in time, following $< k >= at + 2b$. The continuum theory predicts that the time-dependent degree distribution displays a crossover at critical degree

$$k_c = \sqrt{b^2 t}\big(2 + \frac{2at}{b}\big)^{\frac{3}{2}}$$

such that for $k << k_c$, $P(k)$ follows a power law with exponent $\gamma = 1.5$, and for $k >> k_c$ the exponent is $\gamma = 3$. Thus, as time increases, the scaling behaviour with $\gamma = 1.5$ becomes increasingly visible.

There are alternative mechanisms for preferential attachment. For instance, Kumar et al. [2000] proposed the copy model in which new nodes choose an existent node at random and copy a fraction of the links of the existent node. At each time step a new node is added to the network, which connects to the rest of the nodes with a constant number of directed edges. At the same time, a prototype node is randomly chosen from the nodes already in the system. The outgoing edges of the new node are distributed as follows: with probability p the destination of the ith edge is randomly selected, and with probability $1 - p$ it is taken to be the destination of the ith edge of the prototype node. This second process increases the probability of high degree nodes receiving new incoming edges. The authors proved that the expectation of the incoming degree distribution was

$$P(k_{in}) = k^{\frac{-(2-p)}{1-p}}$$

so that $P(k)$ follows a power law with an exponent that varies between 2 (for $p \to 0$) and ∞ (for $p \to 1$).
Following a mathematically similar approach, Krapivsky et al. [2001] proposed a growing network with redirection where at every time step a new node is added to the system and an earlier node i is selected uniformly as a possible target for attachment. With probability $1 - r$ a directed edge from the new node to i is created. But, with probability r the edge is redirected to the ancestor node j of node i. This model generates a power-law distribution with degree exponent $\gamma = 1 + \frac{1}{r}$, which can be tuned to any value larger than 2.

Note that the (stochastic) growth of the networks is not a necessary condition for creating scale-free networks as Dangalchev [2004] proposed deterministic models for creating scale-free networks with given nodes. He built a 2-L model by adding a second order preferential attachment. The activation of a node depends not only on the number of nodes linked to it, but also on the number of links in each of these nodes. That is,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\pi(k_i) = \frac{k_i + C\sum_{i,j} k_j}{\sum_j k_j + C\sum_j k_j^2}$$

where C is a coefficient taking values in $[0, 1]$.
Since the time of birth matters, Hassan et al. [2011] showed that the nodes can be characterised by generalised degree q, the product of the square root of the birth time of each node and their corresponding degree k, instead of the degree k alone. The generalised degree distribution $F(q, t)$ exhibits dynamic scaling

$$F(q, t) \sim t^{-\frac{1}{2}}\phi(\frac{q}{t^{\frac{1}{2}}})$$

where ϕ is the scaling function. It implies that the distinct plots of $F(q, t)$ versus q would collapse into a universal curve if we plots $F(q, t)t^{\frac{1}{2}}$ versus $\frac{q}{t^{\frac{1}{2}}}$. The authors showed that the BA network falls into two universality classes depending on whether new nodes arrive with single edge $(m = 1)$ or with multiple edges $(m > 1)$.

13.3.4 Fractal networks

13.3.4.1 Overview

We have seen in Section (13.3.2) that the small-world properties could be expressed by the slow increase of the average diameter (or average path length) of the network as

$$\ell_{rand} \approx < l > \sim \log n$$

where n is the total number of nodes and l is the shortest distance between two nodes. Equivalently, we can write

$$n \sim e^{\frac{<l>}{l_0}}$$

where l_0 is a characteristic length. Thus, the number of nodes increases exponentially with the diameter of the small-world networks.
In the case of a self-similar structure (see Appendix (24.4.5)), we expect a power-law relation and not an exponential one. Thus, it would seem that small-world networks are not self-similar under a length-scale transformation. However, some authors, such as Song et al. [2005], Moret et al. [2007], analysed a range of real complex networks and showed that they were self-similar on all length scales. This is illustrated in Figure (13.10) with the Sierpinski network and in Figure (13.11) with the Cantor Dust network. Song et al. obtained these results with the fractal scaling approach, which measures a power-law relation between the number of boxes needed to cover the network and the size of the box (see Appendix (27)). These results helps explain the scale-free nature of complex networks and suggest a common self-organisation dynamics. Nowadays, fractal networks help distinguishing global features of complex networks, such as the fractal dimension.

Figure 13.10: Sierpinski fractal networks.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 13.11: Cantor Dust fractal networks.

13.3.4.2 Methods for computing the dimension

Fractal analysis (see Appendix (27)) helps calculating the fractal dimension of complex networks. In order to investigate self-similarity in networks, we can use the box-counting method and renormalisation. Song et al. [2005] considered a renormalisation procedure that coarse-grains the system into boxes containing nodes within a given size. They identified a power-law relation between the number of boxes needed to cover the network and the size of the box, defining a finite self-similar exponent. Kim et al. [2007] introduced a box-covering algorithm as a modification of the approach by Song et al., where fractal networks are viewed as comprising a skeleton and shortcuts.

13.3.4.2.1 The box counting method The method counts the total number of boxes that are needed to cover a network with boxes of certain size (see Appendix (27.3)). Let $N_B(l_B)$ be the number of boxes of linear size l_B needed to cover the given network. The fractal dimension d_B is given by

$$N_B(l_B) \sim l_B^{-d_B}$$

It means that the average number of vertices $< M_B(l_B) >$ within a box of size l_B satisfies

$$< M_B(l_B) > \sim l_B^{d_B} \tag{13.3.5}$$

In order to obtain the fractal dimension d_B, via power-law fit of the distribution, we measure the distribution of N_B for different box sizes or by measuring the distribution of $< M_B(l_B) >$ for different box sizes.

13.3.4.2.2 The cluster growing method The algorithm selects a random node at each step, which is the seed node that will be the center of a box. If the minimum distance l is given, a cluster of nodes separated by at most l from the seed node can be formed. The process is repeated by choosing many seeds until the clusters cover the whole network. Then, the dimension d_f is given by

$$< M_C > \sim l^{d_f}$$

where $< M_C >$ is the average mass of the clusters, defined as the average number of nodes in a cluster. In order to measure the fractal dimension of networks we add the concept of renormalisation.

13.3.4.2.3 Box counting and renormalisation For each size l_B, boxes are chosen randomly (as in the cluster growing method) until the network is covered. A box consists of nodes separated by a distance $l < l_B$. Then, each box is replaced by a node (renormalisation). The renormalised nodes are connected if there is at least one link between the unrenormalised boxes. This process is repeated until the network collapses to one node. Each of these boxes has an effective mass (the number of nodes in the box) which can be used, as described above, to measure the fractal dimension of the network. The networks are also invariant under multiple renormalisations applied for a fixed box size l_B. This invariance suggests that the networks are self-similar on multiple length scales.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

13.3.4.2.4 Skeleton and fractal scaling While the box counting method is embedded in Euclidean space, Euclidean metric is not well defined for networks, so that Kim et al. [2007] developed an adaptation called box-covering algorithm. The fractal properties of the network can be observed in its underlying tree structure. In this view, the network consists of the skeleton and the shortcuts (see Goh et al. [2006], Kim et al. [2007]). The skeleton (within the original network) is a special type of spanning tree, formed by the edges having the highest betweenness centralities [6], and the remaining edges in the network are local shortcuts. In general, the skeleton collects the sections of the network where betweenness is high, which preserves the structure and simplifies its complexity. Thus, if the original network is scale-free, then its skeleton also follows a power-law degree distribution, where the degree can be different from the degree of the original network. For the fractal networks following fractal scaling, each skeleton shows fractal scaling similar to that of the original network. The number of boxes to cover the skeleton is almost identical to that needed to cover the network.
In order to calculate the skeleton of a complex network, we need to calculate the link betweenness of all the links in the network. The betweenness centrality of a network (for a link or a node) is given by

$$b_i = \sum_{j,k \in N, j \neq k} \frac{n_{jk}(i)}{n_{jk}}$$

where N is the total number of nodes, n_{jk} is the total number of shortest-paths between nodes j and k, $n_{jk}(i)$ is the total number of shortest-paths linking nodes j and k that passes through the node i.
Since fractal networks and their skeletons follow the relation in Equation (13.3.5), we can investigate whether a network is fractal and what is its fractal dimension. We can also construct a fractal network by combining a random branching tree and local shortcuts.

13.3.5 Multifractal networks

Some authors, such as Furuya et al. [2011], showed that many real networks could not be described by a single fractal dimension, but were exhibiting multifractal structures (see Appendix (27)). This is illustrated in Figure (13.12). As a result, some algorithms to perform mulifractal analysis on networks developed (see Li et al. [2014]). For instance, Rendon de la Torre et al. [2017] analysed the multifractality of the large scale Estonian network of payments. Recently, researchers proposed several methods for transforming time series into complex networks. Zou et al. [2018] constructed horizontal visibility networks (HVNs) based on the α-stable Levy motion (see Appendix (24.4.4)). They studied the relations of multifractal and Laplacian spectrum of transformed networks on the parameters $1.1 \leqslant \alpha \leqslant 2$ and $|\beta| \leqslant 1$ of the α-stable Levy motion.

[6] A simplified quantity to measure the traffic of networks.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 13.12: Multifractal system.

13.3.5.1 The sandbox algorithm

In order to perform multifractal analysis on networks we need a physical measure, such as the number of nodes within a box of specific size, to analyse how the distribution of such number of nodes scales as the size of the box grows. Following the fixed-size box-counting algorithm (see Halsey et al. [1986]), Tel et al. [1989] considered the box-counting and the sandbox algorithm to determine the generalised dimensions $D(q)$ associated with the geometrical structure of growing deterministic fractals. They showed the two methods were equivalent only if the sandbox method was applied with an averaging over randomly selected centres. In that case the latter outperformed the former in estimating the dimensions. The sandbox method was adapted to multifractal analysis of complex networks by Liu et al. [2015]. The method can be used to calculate the spectrum of the generalised fractal dimensions $D(q)$ and mass exponents $\tau(q)$.

13.3.5.1.1 The fixed size box-counting algorithm We start by describing the fixed size box-counting algorithm for multifractal analysis (see Halsey et al. [1986]) (see Appendix (27.3)). Given a probability measure $0 \leqslant \mu \leqslant 1$ in a metric space Ω with a support set E, for $q \in \mathbb{R}$, we consider the partition sum (see Equation (27.3.10))

$$Z_\epsilon(q) = \sum_{\mu(B)\neq 0} \left[\mu(B)\right]^q$$

describing the moment of the measure. The sum runs over all different non-overlapping (or non-empty) boxes B of a given size ϵ that covers the support set E. Thus, one can easily obtain $Z_\epsilon(q) \geqslant 0$ and $Z_\epsilon(0) = 1$. The function of the mass exponent $\tau(q)$ of the measure μ is defined by (see Equation (27.3.9))

$$\tau(q) = \lim_{\epsilon\to 0}\left(\frac{\ln Z_\epsilon(q)}{\ln \epsilon}\right)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Then, the generalised fractal dimension $D(q)$ of the measure μ are defined as follows (see Equation (27.3.12)):

$$D(q) = \frac{\tau(q)}{q-1}, q \neq 1$$

and

$$D(1) = \lim_{\epsilon \to 0} \frac{Z_{(1,\epsilon)}}{\ln \epsilon}$$

where

$$Z_{(1,\epsilon)} = \sum_{\mu(B) \neq 0} \mu(B) \ln \mu(B)$$

The generalised fractal dimensions $D(q)$ can be estimated with linear regression of $\frac{\ln Z_\epsilon(q)}{(q-1)}$ against $\ln \epsilon$ for $q \neq 1$. Similarly, it can be estimated with a linear regression of $Z_{(1,\epsilon)}$ against $\ln \epsilon$ for $q = 1$. Note, $D(0)$ is the fractal dimension or the box-counting dimension of the support set E of the measure μ, $D(1)$ is the information dimension, and $D(2)$ is the correlation dimension.

13.3.5.1.2 Characterising multifractality For a complex network, a box of size B can be defined in terms of the distance l_B corresponding to the number of links in the shortest-path between two nodes. It means that every node is less than l_B links away from another node in the same box. The measure μ of each box is defined as the ratio of the number of nodes that are covered bu the box and the total number of nodes in the whole network.
Multifractality of a a complex network is determined by the shape of $\tau(q)$ or $D(q)$ curves. If $\tau(q)$ is a straight line, or $D(q)$ is a constant, then the network is monofractal. Otherwise, if $\tau(q)$ or $D(q)$ have convex shapes, then the network is multifractal. In general, mulifractal structure can be defined as

- multiple slopes of $\tau(q)$ versus q
- non-constant $D(q)$ versus q values
- $f(\alpha)$ versus α covers a broad range (not accumulated at nearby non-integer values of α)

13.3.5.1.3 The algorithm The main idea of the sandbox algorithm is to select a node of the network in a random fashion as the center of a sandbox and then count the number of nodes that are inside the sandbox. As an example, we described the algorithm introduced by Rendon de la Torre et al. [2017].
Start by calculating the shortest-path distance between any two nodes in the network and map the shortest-path adjacency matrix $B_{N \times N}$ using the payments adjacency matrix $A_{N \times N}$. The matrix $B_{N \times N}$ is used as input in the multifractal analysis. At inception, no node has been chosen as a center of a box. The radius r of the sandbox is in the range $[1, d]$, where the diameter d is the longest distance between nodes in the network and radii r are integer numbers. The nodes are randomly reordered in the whole network. Depending on the size N of the network, the authors chose T nodes in random order as centers of T sandboxes. All the neighbouring nodes within radius r from the center of each box are found. The number of nodes contained in each sandbox of radius r is counted, and denoted by $S(r)$. Also, $S(0)$ is the size of the network. Then, one calculate the statistical averages $< [S^{q-1}(r)] >$ of $[S^{q-1}(r)]$ over all the sandboxes T of radius r. The previous step is repeated for each of the different values of radius r to obtain the statistical average $< [S^{q-1}(r)] >$. It is then used to calculate the linear regression. The generalised fractal dimensions $D(q)$ of the measure μ are defined by

$$D(q) = \lim_{r \to 0} \frac{\ln < [\frac{S(r)}{S(0)}]^{q-1} >}{\ln \frac{r}{d}} \frac{1}{q-1}, q \in \mathbb{R}$$

which can be written as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\ln < [S(r)]^{q-1} > \propto D(q)(q-1)\ln\frac{r}{d} + (q-1)\ln S(0)$$

The generalised fractal dimensions $D(q)$ is obtained by running the linear regression of $\ln < [S(r)]^{q-1} >$ versus $(q-1)\ln\frac{r}{d}$. Similarly, the mass exponents $\tau(q)$ are obtained by calculating the linear regression of $\ln < [S(r)]^{q-1} >$ versus $\ln\frac{r}{d}$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 14

Neural NLP

We introduce representation learning and deep neural network-style machine learning methods (see Chapter (9)) within natural language processing (NLP), and show how they achieved state-of-the-art results in many natural language tasks, such as language modelling, parsing, and many others.

14.1 Sequence to sequence

14.1.1 Description

14.1.1.1 Overview

14.1.1.1.1 Seq2Seq model Sequence to sequence (Seq2Seq) turns (or transforms) one sequence into another sequence by using a recurrent neural network (RNN) (see Section (6.1)), or more often LSTM or GRU, to avoid the problem of vanishing gradient (see Sutskever et al. [2014]). The two sequences can be of the same, or arbitrary, length. Similarly to generating sequences with RNNs in Section (12.6.2.2), the context for each item is the output from the previous step, but the target is a different sequence. It is primarily used for translation (statistical machine translation (SMT) and neural machine translation (NMT)), text generation, and image captioning. For instance, Schwenk [2012] proposed to use feedforward neural networks in the framework of phrase-based SMT system. Kalchbrenner et al. [2013] successfully used neural networks as a sub-component of the existing translation system. Rather than using a neural network as a part of the existing system, NMT models such as Bahdanau et al. [2015], Luong et al. [2015], work on their own and generates a translation from a source sentence directly.
A typical seq2seq network consists of a bidirectional encoder, a unidirectional decoder, and an attention mechanism between the encoder and the decoder. The encoder-decoder system is as follows:

- One encoder network: The encoder neural network reads and encodes a source sentence into a fixed-length vector. It turns each item into a corresponding hidden vector containing the item and its context.
- One decoder network: The decoder outputs another sentence or a translation from the encoded vector. It reverses the process, turning the vector into an output item, using the previous output as the input context.

The whole encoder-decoder system is jointly trained to maximise the probability of a correct sentence or translation given a source sentence.

We let $(x, y) \in (\mathcal{X}, \mathcal{Y})$ be a sentence pair where $\mathcal{X}$ and $\mathcal{Y}$ are the source and target domain, respectively. We let $x = (x_1, ..., x_n) \in \mathcal{X}^{T_n}$, for some domain $\mathcal{D}$ and time length T_n (also denoted T_x), be the source sentence with n tokens, and $y = (y_1, ..., y_m) \in \mathcal{Y}^{T_m}$, for the time length T_m (also denoted T_y), be the target sentence with m tokens. From a probabilistic approach, we want to find a target sentence y that maximises the conditional probability

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

of y given a source sentence x, that is, $\arg\max_y P(y|\, x)$. This means that the sentence predicted most often by the model is the final sentence. Thus, a sequence to sequence model learns the model parameter θ to estimate the conditional probability $P(y|\, x;\theta)$. Once the conditional distribution is learned by the model, given a source sentence, a corresponding sentence or translation can be generated by searching for the sentence that maximises the conditional probability. This is obtained with an encoder-decoder model as illustrated in Figure (14.1).

Figure 14.1: Seq2Seq model.

The elements of the input sequence $x = \{x_1, x_2, ..., x_n\}$, called tokens [1], can be literally anything from text representations, to pixels, or even images in the case of videos. This mechanism aims at transforming an input sequence (source) to a new one, called the target, by encoding a whole input sentence into a fixed-length vector, z, from which a target sentence or translation will be decoded.
This approach has been extended over time, leading to improved Seq2Seq models. Some of the additional components include:

- Attention: The input to the decoder is a single vector which stores the entire context. Attention allows the decoder to look at the input sequence selectively.
- Beam Search: Instead of picking the single output (word) as the output, multiple highly probable choices are retained, structured as a tree (using a Softmax on the set of attention scores). Average the encoder states weighted by the attention distribution.
- Bucketing: Variable-length sequences are possible because of padding with 0s, which may be done to both input and output. However, if the sequence length is 100 and the input is just 3 items long, expensive space is wasted. Buckets can be of varying sizes and specify both input and output lengths.

14.1.1.1.2 The encoder and decoder The encoder-decoder models are nothing more than stacked RNN layers, such as LSTM layers (see Figure (12.13)). They are used to encode a variable-length input sentence x into a fixed-length vector z which serves to predict the target sentence or translation y. Examples include the RNN Encoder-Decoder proposed by Cho et al. [2014] and Sutskever et al. [2014].

- The encoder processes the input sequence x (each item at a time) and produces one compact representation, called **context vector** and denoted z, from all the input time steps. It can be regarded as a compressed format of the input sequence. Following Section (12.6.1), we let $h = \{h_1, h_2, ..., h_n\}$ be the state variables (or latent variables in presence of hidden states) of the system. Here, $h_i \in \mathbb{R}^d$, $i = 1, ..., n$, is the state variable for the ith token. The dynamics of the state-space model in the case of a single hidden layer is given in Equation (12.6.16). In our setting, a RNN takes two inputs at each time step: an input (one vectorised word from the input sentence), and a hidden state, and it produces two output vectors. In the one hidden layer framework, we get

$$h_i = f(x_i, h_{i-1})$$

and

[1] In NLP, a token is a string with an assigned and identified meaning. A word token is a word observed in a piece of text

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$z = f_z(h)$$

where f and f_z are some nonlinear functions. For example, Sutskever et al. [2014] let f be an LSTM, and $f_z(h) = h_n$ be the last hidden state of the sequence. The output of the RNN block is the input value of the next RNN block until the last block, where the output is the context vector. Using graph theory, the computational graph of the encoder is illustrated in Figure (14.2).

Figure 14.2: Encoder model.

- On the other hand, the decoder receives the context vector z as input and generates the output sequence y (item by item) with a set of RNNs. To do so, it maintains a hidden states, denoted h_j or s_j, that it passes from one time step to the next. For example, in language translation, we can think of the input sequence as the representation of a sentence in English and the output as the same sentence in French.
  Practically, the decoder is trained to predict the next value y_i (word, pixel, image) given the context vector z and all the previously predicted values $\{y_1, ..., y_{i-1}\}$. It defines a probability over the sentence or translation y by decomposing the joint probability into the ordered conditionals

$$P(y) = \prod_{i=1}^{T_y} P(y_i | \, y_1, ..., y_{i-1}; z)$$

  where $y = (y_1, ..., y_{T_y})$. With an RNN model, each conditional probability is modelled as

$$P(y_i | \, y_1, ..., y_{i-1}; z) = g(y_{i-1}, s_i, z) \tag{14.1.1}$$

  where g is a nonlinear, potentially multi-layered, function that outputs the probability of y_i, and s_i (or h_i if we use the same notation as in the encoder) is the hidden state of the RNN decoder. The computational graph of the decoder is illustrated in Figure (14.3).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 14.3: Encoder model.

14.1.1.2 Attention mechanisms

The context vector z needs to capture all the information about the source sentence. In practice, z cannot encode information from all the input time steps, making it challenging for the model to deal with long sentences. This is commonly known as the *bottleneck problem*. As discussed in Section (6.1), RNNs tend to forget information from time steps that are far behind. Furthermore, the stacked RNN layer usually create the well-know vanishing gradient problem (see Section (6.1.3)).

Attention mechanisms (see Bahdanau et al. [2015], Luong et al. [2015]) were born in order to address these two well known problems of the Seq2seq model. Attention allows the model to focus on the relevant parts of the input sequence as needed by letting the context vector z have access to all parts of the input sequence instead of just the last one. We can look at all the different words at the same time and learn to "pay attention" to the correct ones depending on the task at hand. In general, the encoder passes a weighted average of all the hidden states to the decoder. Hence, attention is simply a notion of memory through time. When we assume that the attention is computed over the entire input sequence $x = (x_1, ..., x_{T_x})$, it is called *global attention*.

14.1.1.2.1 Description The attention extend the encoder-decoder model described above by letting a model (soft or hard-) search for a set of input words, or their annotations computed by an encoder, when generating each target word y_i. This way, the model does not have to encode a whole source sentence into a fixed-length vector, and can focus only on information relevant to the generation of the next target word. The whole system is jointly trained towards a better log-probability of producing better accuracy.
Given the previous state in the decoder, s_{i-1}, where i indicates the prediction step (or simply time step), and the hidden states in the encoder, $h = \{h_1, h_2, ..., h_n\}$, we have

$$e_i = A_{net}(s_{i-1}, h) \in \mathbb{R}^n \tag{14.1.2}$$

where $A_{net}(\cdot, \cdot; W)$ is an attention network with weight matrix W. Here, the attention (or alignment) at time step i, denoted e_i, is the output of a small neural network with inputs the previous state of the decoder, s_{i-1}, and the hidden state h.
All we need is a score that describes the relationship between the two states of the encoder and decoder, and captures the way they are aligned. Several approaches have been proposed to define a score between the hidden state of the decoder, s_{i-1}, and all the hidden states of the encoder h. We choose to present the softmax method. At each time step i, and for each hidden state h_j, $j = 1, ..., n$, we calculate a scalar:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$e_{ij} = A_{net}(s_{i-1}, h_j) \, , i = 1, ..., m \tag{14.1.3}$$

which scores how well the inputs around position j and the output at position i match. The score is based on the hidden state s_{i-1} (just before emitting y_i) and the jth hidden state h_j of the input sentence. In this setting, we compute the annotation weight of every word in the source sentence for each word in the target sentence or translation.
Further, we choose to get some extra properties:

1. Make the score a probability distribution.
2. Make the scores e_{ij} be far from each other (the s_{i-1} and h_j hidden states).

Thus, we let the weight α_{ij}, computed at each time step i, for each state h_j, be given by the Softmax function (see Equation (10.1.4))

$$\alpha_{ij} = \frac{e^{e_{ij}}}{\sum_{k=1}^{T_x} e^{e_{ik}}} \, , i = 1, ..., m \tag{14.1.4}$$

Note, in that setting, the probability in Equation (14.1.1) is conditioned on a distinct context vector z_i for each target word y_i. Thus, we get the set $z = \{z_1, ..., z_m\}$. Here, the context vector at the ith time step (in the decoder) is given by

$$z_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \, , i = 1, ..., m \tag{14.1.5}$$

Since $h_j \in \mathbb{R}^d$, all the dimensions of h_j are multiplied by α_{ij} in the calculation of z, and the sum is computed elementwise. If we denote α a $n \times m$ matrix, we can rewrite the above system as

$$z = \sum \alpha^\top \bigotimes h$$

where $\bigotimes$ is the elementwise multiplication and the sum is computed elementwise in the time direction of the encoder. Then, we obtain the matrix z of size $m \times d$. Hence, the weight α_{ij} is a data-dependent dynamic weight, so that the weighting is a learned function. Note, the context vectors z_i, $i = 1, .., m$, are then used to compute a new state sequence s in the decoder, where each state s_i depends on s_{i-1}, z_i and y_{i-1}, the model's output at time step $i - 1$. In the special case where, s_i, is a RNN hidden state for time i, it is mathematically written as

$$s_i = f(s_{i-1}, y_{i-1}, z_i) \tag{14.1.6}$$

for some nonlinear function f. This is then used to compute the conditional probability in Equation (14.1.1). This attention mechanism is displayed in Figure (14.4).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 14.4: Attention mechanism.

14.1.1.2.2 Scoring the states In order to compute the above attention, we just need to define a score, S, that describes the relationship between the two states of the encoder and decoder, and captures how they are aligned. One simple solution is to use the fact that the product of two vectors gives a measure of their similarity. The higher the dot product, the higher the similarity. Different approaches have been proposed for the score value. For instance, Luong et al. [2015] computed attention as the dot product between the two states $s_{i-1} \cdot h$. Extending this idea we can introduce a trainable weight matrix between the two states, getting $s_{i-1} W_a h$, where W_a is an intermediate matrix with learnable weights. Alternatively, we can choose a particular network A_{net}. For example, Bahdanau et al. [2015] proposed

$$S(h, s_{i-1}) = v_a^\top \tanh\big(W_a[h, s_{i-1}]\big)$$

where v_a is another weight to estimate. Graves et al. [2014] calculated the attention as a cosine similarity

$$S(h, s_{i-1}) = \cos[s_{i-1}, h]$$

Note, despite its simplicity, the attention can be computationally expensive and sometimes unnecessary. As a result, some authors proposed local attention as a solution. In local attention, we consider only a subset of the input

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

units/tokens. It can also be seen as hard attention since we need to take a hard decision, by first excluding some input units. This is illustrated in Figure (14.5).

Figure 14.5: Global (LHS) and local (RHS) attention. The colours in the attention indicate constantly changing weights.

Self-attention is the key component of the Transformer architecture. It is the attention of the same sequence with the goal of creating a meaningful representation of the sequence before transforming it to another one. Rather than looking for an input-output sequence association/alignment, we are now looking for scores between the elements of the same sequence. It can be regarded as a (k-vertex) connected undirected weighted graph [2]. This illustrated as a probability score matrix and undirected weighted graph in Figure (14.6).

	Hello	I	love	you
Hello	0.8	0.1	0.05	0.05
I	0.1	0.6	0.2	0.1
love	0.05	0.2	0.65	0.1
you	0.2	0.1	0.1	0.6

Figure 14.6: Self-attention mechanism.

Even though Transformers are used for machine translation, they are often considered as general-purpose NLP models, effective on tasks like text generation, chatbots, text classification, etc.

14.1.1.3 Some simplified attentions

14.1.1.3.1 A simple and improved attention Raffel et al. [2016] proposed a simplification to the above attention mechanism, allowing it to be used to produce a single vector z from an entire input sequence $x = (x_1, ..., x_n)$. Note, there is no decoder in that setting. It corresponds to the encoder described in Section (14.1.1.1), but with an improved context vector defined as the weighted average of the state sequence h. At each time step i, the simplified attention is formulated as follows:

$$e_i = A_{net}(h_i) \, , \, \alpha_i = \frac{e^{e_i}}{\sum_{k=1}^{T_x} e_k} \, , \, z = \sum_{i=1}^{T_x} \alpha_i h_i \tag{14.1.7}$$

where $A_{net}(\cdot; W)$ is a learnable function with weight matrix W, which only depends on the hidden state h_i at time step i (as opposed to using all the states h_j for $j = 1, ..., n$ for every time step i). In general, A_{net} is set to $\tanh$. This simplified attention is illustrated in Figure (14.7).

[2] Undirected indicates that the matrix is symmetric.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 14.7: Simplified attention mechanism where c is the context vector and a is the learnable function.

When it comes to implementing this attention, since $h_i \in \mathbb{R}^d$, all the dimensions of $h_i = (h_{i,1}, ..., h_{i,d})$ are multiplied by $\alpha_i = (\alpha_{i,1}, ..., \alpha_{i,d})$ in the calculation of z, and the sum is computed elementwise. That is, if we denote α the matrix of size (d, n) with columns the vectors α_i, $i = 1, ..., n$, and let the matrix h have size (d, n) (taking the transpose), we can rewrite the above system as

$$z = \sum \alpha \bigotimes h$$

where $\bigotimes$ is the elementwise multiplication and the sum is computed elementwise in the time direction. Then, we obtain a vector z of dimension d. This is illustrated in Figure (14.8).

Remark 14.1.1 *In this formulation, the attention can be seen as producing a fixed-length embedding z of the input sequence by computing an adaptive weighted average of the state sequence h. This is an improvement to the fixed-length vector z proposed by Cho et al. [2014] and Sutskever et al. [2014], which is the last hidden state of the sequence h_n. This is also an improvement to the unweighted average of the state sequence $z = \frac{1}{T_x}\sum_{i=1}^{T_x} h_i$ used by Dieleman [2014] to collapse the temporal dimension of audio, and Lei et al. [2015] for text document (see Section (14.2.1)).*

Remark 14.1.2 *This model will fail on any task where temporal order matters since computing an average over time discards order information. The authors argued that temporal order was substantially less important than being able to handle very long sequences.*

Remark 14.1.3 *It seems that the task in this article was not to produce a target sequence y from an input sequence x. The authors focused on the addition and multiplication problems (see Hochreiter et al. [1997]). Thus, there is no-longer a need for an alignment between the hidden states of the encoder and those of the decoder.*

One consequence of this simplified form of attention is that a model can handle variable-length sequences even if the calculation of h_i was feed-forward, that is, $h_i = f(x_i)$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 14.8: Implementation of the simplified attention mechanism where z is the context vector.

14.1.1.3.2 A hierarchical attention network Yang et al. [2015] proposed a hierarchical attention network for document classification with two distinctive characteristics:

1. It has a hierarchical structure that mirrors the hierarchical structure of documents.
2. It has two levels of attention mechanisms applied at the word and sentence-level, enabling it to attend differentially to more and less important content when constructing the document representation.

Here, [2] let the model pay more or less attention to individual words and sentences when constructing the representation of the document. This is achieved by introducing a new variable, the context, where the most relevant information is concentrated. The model consists of several parts: a word sequence encoder, a word-level attention layer, a sentence encoder and a sentence-level attention layer.
In this section, we focus on the sentence attention. Given the sentence vectors s_i, the authors used a bidirectional GRU to encode the sentences, getting the annotation of the ith sentence as $h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}]$. Then, they used an attention mechanism and introduced a sentence level context vector u_s to measure the importance of the sentences. The mechanism is as follows:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$
\begin{aligned}
u_i &= f_u\big(W_s h_i + b_s\big) \\
\alpha_i &= \frac{e^{u_i^\top u_s}}{\sum_k e^{u_k^\top u_s}} \\
v &= \sum_i \alpha_i h_i
\end{aligned}
$$

where $f_u = \tanh$, and v is the document vector summarising all the information of sentences in a document. They let the additional feedforward layer determine the value of this context along with the weights and bias during training.

To summarise the above two approaches:

- The first approach generates a set of attention weights, use it to calculate the attention-adjusted state, and use this for prediction.
- The second approach generates a context, use it to generate a set of attention weights, and use this to calculate the attention-adjusted state and make prediction.

Practically, we can use the previous hidden state as a pseudo-context, using it to generate a set of attention weights, and use this to calculate the attention-adjusted state. Then, we concatenate the previous hidden state, the attention-adjusted state and the translated word and use this for prediction

14.1.1.3.3 Implementation We choose to illustrate the above simplified attention mechanism in a classification problem, and give the implementation of a self attention layer in Python using Keras. In this example, we are interested in taking a text from social media and assessing whether the sentiment of that text is positive or negative. When excluding the attention mechanism, we feed the texts to an RNN model (LSTM or GRU), which retains a memory of the sequence, and then use the final hidden state as an input to a feedforward dense neural net layer which learns and then classifies the texts. This is a special case of the encoder-decoder described in Section (14.1.1), where the single context vector is the output after the last word, and the decoder is a classifier returning the sentiment of a text. To capture information on the whole sequence, we add an attention mechanism and consider a state-space model with a bidirectional LSTM/GRU and a deep network as classifier.

We let the input be a set of texts $\mathcal{T} = \{t_1, t_2, ..., t_q\}$, where each text t_i contains T_n words (the tokens). We use one layer of embedding to convert each word in each text into a 100 dimensional vector, leading to a $T_n \times 100$ dimensional vectors for each text. Hence, the sequence length (or the number of time-steps) of the RNN has size T_n. We denote `seq_length` the sequence length in the code. We let the bidirectional LSTM/GRU encoder have N_U output units generating one hidden state h_i for each word W_i, for $i = 1, ..., n$. Thus, the number of dimensions for each hidden state becomes $N_D = 2 \times N_U$.
In the classical RNN encoder, only the last word's hidden state h_n is returned as the output for each text. That is, the output (or context vector) is a single vector of size N_D for each text. This sentence embedding is passed as input to a feedforward dense layer that classifies texts and return their sentiment. Here, it returns one binary value for each text. Rather than just using the last hidden state of the LSTM as an input to the feedforward layer, we choose to pay attention to every single hidden state generated by adding an attention mechanism. To do so, we first need to decide which ones are important, and then take a weighted sum of hidden states (words) and feed this as input to the deep network. In the code, we use the `return_sequences = true` option (it returns a 3D shape object) in the LSTM layer to return a hidden state for every word within the text (the sequence length), leading to an output vector of T_n hidden states with N_D dimensions. Thus, we consider a three-dimension tensor object with shape denoted by (`batch_size`, `time_steps`, `input_dim`), where `time_steps` is the length of the sequence, and `input_dim` the number of dimensions.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The attention mechanism assigns one weight per word, getting a total of T_n weights, which we choose to estimate with a neural network. Thus, we need to create an additional simple feedforward layer to estimate these weights. Since this task is a linear problem, we let that layer be a single dense layer Multilayer Perceptron (MLP) with a single output unit. The number of attention weights is the sequence length of the RNN (or the number of time-steps) corresponding to the number of words in the text. As a result, the shape of the output should be $(\cdot, T_n, 1)$. Note, this layer also needs to return the weighted sum of words (the context z), which becomes the input to the next layer in the deep network. It is called the *attention adjusted output* state with shape given by $(\cdot, 1, N_D)$. Thus, the input-output of that layer are as follows:

- The input consists of all the T_n hidden states of the LSTM sequence from the previous step. Since each hidden state has size N_D, the input shape should be $(\cdot, T_n, N_D)$.
- The output is a set of attention weights with shape $(\cdot, T_n, 1)$, one per each word in the sequence, together with the attention adjusted output state with shape $(\cdot, 1, N_D)$.

The main idea behind the attention mechanism is to use the attention weights to magnify or diminish each word in order to create the final word (the context vector z). Mathematically, we multiply (matrix multiplication) the input shape $(\cdot, T_n, N_D)$ with the dimensions $(\cdot, N_D, 1)$, yielding a set of attention weights with shape $(\cdot, T_n, 1)$. We then perform a simple multiplication to get the weighted sum, which is fed into the deep network. We multiply the N_D dimensions of each of the ith hidden state h_i by the attention weight a_i, for all i, getting the final word (see Equation (14.5.9)). In the code, we repeat N_D times the attention weight $(\cdot, T_n, 1)$, getting the shape $(\cdot, N_D, T_n)$, which is permuted to get the shape $(\cdot, T_n, N_D)$. We can then compute the Hadamard product (element-wise multiplication) between that shape and the input shape $(\cdot, T_n, N_D)$, and take the sum, obtaining the output $z = a_1 \times h_1 + a_2 \times h_2 + \cdots + a_n \times h_n$, which is a vector of dimension N_D. Note, the attention weights are not fed into the next layer.

We now present some simple code in Python using Keras for the implementation of the self attention layer described above. The code for the self attention layer is given below:

```
class SelfAttLayer(Layer):
    def __init__(self, **kwargs):
        self.attention = None
        self.init = initializers.get('normal')
        super(SelfAttLayer, self).__init__(**kwargs)


    def build(self, input_shape):
        assert len(input_shape) == 3
        self.inp_dimensions = input_shape[-1]
        self.seq_length = input_shape[-2]
        num_units = 1
        self.W = self.init((self.inp_dimensions, num_units))
        self.b = self.init((self.seq_length, num_units))
        super(SelfAttLayer, self).build(input_shape)

    def call(self, x, mask=None):
        e = K.tanh(K.dot(x,self.W)+self.b)
        e = Flatten()(e)
        a = Activation('softmax')(e)
        temp = RepeatVector(self.inp_dimensions)(a)
        temp = Permute([2,1])(temp)
        output = Multiply()([x,temp])
```


This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

```
        result = Lambda(lambda values: K.sum(values, axis=1))(output)

        return result
```

14.1.1.3.4 Application to language translator As discussed in Section (14.1.1.2), in the case of language translator, Bahdanau et al. [2015] and Luong et al. [2015] proposed a specific attention mechanism. Rather than encoding a whole input sentence into a single fixed-length vector x, it encodes the input sentence into a sequence of context vectors z_i, $i = 1, .., m$, and chooses a subset of these vectors adaptively while decoding the translation. We choose to revisit this mechanism and give a simple implementation.
At the first step of the decoder, the first hidden state of the decoder (a pseudo context) holds the essence of the input sentence. As the decoder proceeds to translate word by word, the decoder also outputs an updated hidden state, s_i, at each time step i, which becomes the pseudo context vector for the next step (see Equation (14.1.6)). An alignment score is computed on the hidden states of the encoder and decoder (see Equation (14.1.3)), which is softmaxed to give the attention weights (see Equation (14.1.4)). The latter are multiplied with the encoder hidden states, h, and summed up to produce the context vector z_i, or, attention-adjusted output state (see Equation (14.1.5)). Then, feed the hidden state s_i, the attention-adjusted state z_i, and the previous translated word y_{i-1} in a network, and predict the translated word y_i (see Equation (14.1.8)). The output of the RNN, at time i, is the next translated word y_i and a new hidden state s_i.
Practically, the RNN takes the value of the last hidden state, h_n, from the encoder and return the translation for the first word y_1. In order to translate the second word x_2, it uses the hidden state s_1 generated by the decoder in the previous time-step and also the just-translated word y_1, and generates the second word y_2 as well as a second hidden state s_2. Similarly, the ith translated word y_i is generated with two inputs: (1) the previous hidden state s_{i-1}, (2) the previous translated word y_{i-1}.
The authors recommend the value of the hidden state, s_{i-1}, to be uni-directional, whiles the values of the states h_j, $j = 1, .., n$, are bi-directional. Note, some variations have been proposed to the way the context vector is calculated. For example, rather than computing the dot product between the hidden states of the encoder and decoder, some authors use a single layer MLP to find three separate weights u, v, w such that the attention weights are the softmax of

$$v \times \big(w \times s_{i-1} + u \times h\big)$$

In that setting, the previous hidden state s_{i-1} is weighted by w, the encoder hidden states h is weighted by u, and the sum of these two terms is weighted by v. Note, the authors used more than one unit in the MLP to determine the weights w and u. The weights that are applied individually on the decoder hidden state at $i-1$ and the encoder hidden states. The additional weight v is used to transform the tensor shape back to $(\cdot, T_n, 1)$. Determining v is a simple linear transformation just requiring one unit. The code for this language translator is given below:

```
class Translator(Layer):
    def __init__(self, n):
        super(Translator, self).__init__()
        self.w = Dense(n)
        self.u = Dense(n)
        self.v = Dense(1)

    def call(self, seq_s, h):
        seq_s = expand_dims(seq_s, 1)
        e = self.v(tanh(self.w(seq_s) + self.u(h)))
        a = softmax(e, axis=1)
        output = a * h
        result = reduce_sum(output, axis=1)
```


This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

```
        return result
```

14.1.2 Modelling Seq2Seq

14.1.2.1 Introduction

14.1.2.1.1 A brief history As discussed in Chapters (5) and (6), deep neural networks (DNNs) and recurrent neural networks (RNNs) are powerful machine learning models, as they can perform arbitrary parallel computation for a small number of steps. In general, large DNNs are trained with supervised backpropagation whenever the labelled training set has enough information to specify the meta parameters. In addition, DNNs can only be applied to problems whose inputs and output targets can be sensibly encoded with vectors of fixed dimensionality. However, many practical problems are best expressed with sequences whose lengths are not known a-priori, such as speech recognition and machine translation (see Section (1.5)). Similarly, question answering tasks maps a sequence of words representing the question to a sequence of words representing the answer (see Section (1.5.2)). Thus, these problems need a domain-independent method that learns to map sequences to sequences.

These types of problems experience considerable time lag between the inputs and their corresponding outputs, such that the data exhibits long range temporal dependencies. As discussed in Section (12.6.1), recurrent neural networks (RNNs) are sequential, generating a sequence of hidden states h_t, as a function of the previous hidden state h_{t-1} and the input for position t. As a result, we should consider RNN models when learning this data. RNNs require that the dimensionality of the inputs and outputs is known and fixed. Thus, the RNN can easily map sequences to sequences whenever the **alignment** between the inputs and the outputs is known ahead of time. However, it is more difficult to apply an RNN to problems whose input and the output sequences have different lengths with complicated and non-monotonic relationships.

To solve this problem, Kalchbrenner et al. [2013] considered continuous representations for words by mapping the entire input sentence from the source language to vector. Graves [2013] introduced a novel differentiable attention mechanism allowing neural networks to focus on different parts of their input. He showed how LSTM recurrent neural networks could be used to generate complex sequences with long-range structure, simply by predicting one data point at a time. The approach was demonstrated for text (where the data are discrete) and online handwriting (where the data are real-valued).

Recent works considered using one RNN for reading the input sequence, one time step at a time, to obtain large fixed-dimensional vector representation (context), and then use another RNN to extract the output sequence from that vector (see Section (14.1.1)). The second RNN is a language model (see Mikolov et al. [2010], Sundermeyer et al. [2010]), but also conditioned on the input sequence. For instance, Cho et al. [2014] proposed the RNN Encoder-Decoder, which combines two models in an encoder-decoder process:

1. Encoder: to map a variable-length input sequence to a fixed-sized vector representation using one RNN, and then
2. Decoder: to map the (fixed-length) vector representation to a variable-length target sequence with another RNN.

The encoder and decoder are jointly trained to maximise the conditional probability of the target sequence given a source sequence. Further, they use a sophisticated hidden unit to improve both the memory capacity and the ease of training. Following the same approach, Sutskever et al. [2014] proposed a two steps process:

1. use an LSTM to read the input sequence, one time-step at a time, to obtain large fixed dimensional vector representation, and
2. use another LSTM to extract the output sequence from that vector.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In that setting, the second model is conditioned on the input sequence. The authors chose the LSTM model since it is efficient at learning problems with long range temporal dependencies.
Ramachandran et al. [2016] proposed to train two independent language models for the encoder and decoder, respectively.

As discussed in Section (14.1.1.1), attention mechanisms allow the modelling of dependencies, even if they are distant in the input or output sequences. These attention mechanisms are used with recurrent networks (see Bahdanau et al. [2015], Kim et al. [2017]). Bahdanau et al. [2015] proposed a new architecture consisting of a bidirectional RNN as an encoder and a decoder that emulates searching through a source sentence during decoding a translation. Vaswani et al. [2017] proposed a simple network architecture, called the Transformer, solely based on attention mechanism, with no recurrence and convolutions. Results on translation tasks showed superior in quality.

14.1.2.1.2 Settings Following Section (12.6.2), on the generation of sequences in the LSTM model, we present the basic framework of sequence to sequence learning. In that setting, we let the input and output sequences be sentences. We let $(x, y) \in (\mathcal{X}, \mathcal{Y})$ be a sentence pair where $\mathcal{X}$ and $\mathcal{Y}$ are the source and target domain, respectively. We let $x = (x_1, ..., x_n)$ be the source sentence with n tokens, and $y = (y_1, ..., y_m)$ be the target sentence with m tokens. A sequence to sequence model learns the model parameter θ to estimate the conditional probability $P(y|\, x; \theta)$. Given the chain rule, it can be further factorised to

$$P(y|\, x; \theta) = \prod_{t=1}^{m} P(y_t|\, y_{<t}, x; \theta)$$

where $y_{<t}$ is the proceeding tokens before position t. The objective function is the log-likelihood

$$\mathcal{L}(\theta; (\mathcal{X}, \mathcal{Y})) = \sum_{(x,y)\in(\mathcal{X},\mathcal{Y})} \log P(y|\, x; \theta)$$

We can then sample a new sequence from this learned distribution. We are now going to detail some of the implementations of this conditional probability.

14.1.2.2 The recurrent language model

Kalchbrenner et al. [2013] proposed the RCTM (recurrent continuous translation model) to estimate the conditional probability $P(y|\, x)$ of a target sentence $y = \{y_1, ..., y_m\}$ being a translation of a source sentence $x = \{x_1, ..., x_n\}$. Given $y_{i:j}$ a substring of words $y_i, ..., y_j$, then the conditional probability of the sequence y satisfies

$$P(y|\, x) = \prod_{i=1}^{m} P(y_i|\, y_{1:i-1}, x)$$

where $P(y_i|\, y_{1:i-1}, x)$ is the conditional probability of the target word y_i occurring in the translation at position i, given the preceding target words $y_{1:i-1}$ and the source sentence x. Thus, we need:

1. a generative architecture for the target sentence y.
2. an architecture for conditioning the latter on the source sentence x.

Here, the model has (1) a generation of translation modelled with a target recurrent language model (RLM) based on recurrent neural network (RNN), (2) a conditioning on the source sentence modelled with a convolution sentence model. Both (1) and (2) above are trained as a single joint architecture.
To understand the RCTM, we briefly describe the recurrent language model (RLM). In the RLM, we want to compute the probability, $P(y)$, that the sequence of words y occurs in a given language, which is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$P(y) = \prod_{i=1}^{m} P(y_i|\, y_{1:i-1})$$

where $P(y_i|\, y_{1:i-1})$ is the conditional probability of the target word y_i occurring in the translation at position i, given the preceding target words $y_{1:i-1}$. We consider a vocabulary V containing the words y_i as well as three transformations:

- an input vocabulary transformation $I \in \mathbb{R}^{q\times|V|}$
- a recurrent transformation $R \in \mathbb{R}^{q\times q}$
- an output vocabulary transformation $O \in \mathbb{R}^{|V|\times q}$

For each word $y_k \in V$, we denote $i(y_k)$ its index in V and $v(y_k) \in \mathbb{R}^{|V|\times 1}$ all zero vector with only $v(y_k)_{i(y_k)} = 1$. For a word y_i, then the result of $I \cdot v(y_i) \in \mathbb{R}^{q\times 1}$ is the input continuous representation of y_i, and q governs the size of the word representation. In the LSTM, the computation of each $P(y_i|\, y_{1:i-1})$ is done recursively via Equation (12.6.17). For $1 < i < m$, we get

$$\begin{aligned} h_1 &= \sigma\big(I \cdot v(y_1)\big) \\ h_{i+1} &= \sigma\big(R \cdot h_i + I \cdot v(y_{i+1})\big) \\ o_{i+1} &= O \cdot h_i \end{aligned}$$

where σ is a non-linear function such as $\tanh$ and $I \cdot v(y_{i+1})$ is the input continuous representation of y_{i+1}. As discussed in Statement (12.6.1), we need to choose the form of the conditional distribution to match the input data. Here, it is given by the Softmax function (see Equation (10.1.4))

$$P(y_i = v|\, y_{1:i-1}) = \frac{e^{o_{i,v}}}{\sum_{v=1}^{V} e^{o_{i,v}}}$$

The RLM is trained by backpropagation through time. The RCTM can be seen as the RLM in which the predicted distributions for each word y_i are conditioned on the source sentence x.

14.1.2.3 The encoder-decoder process

In the case of the encoder-decoder framework (see Section (14.1.1.1)):

- the encoder reads a variable-length source sequence x and generates a set of representations z (or v);
- the decoder estimates the conditional probability of each target token y_t given the source representations and its preceding tokens y_{t-1}.

Sutskever et al. [2014] and Cho et al. [2014] proposed the RNN Encoder-Decoder, which combines two models in an encoder-decoder process. It is a statistical machine translation system consisting of an encoder-decoder model made with two recurrent neural networks (RNN):

1. One RNN encodes a sequence of symbols into a fixed length vector representation.
2. The other decodes the representation into another sequence of symbols.

This system is jointly trained to maximise the conditional probability P of a target sequence given a source sequence. Then, Bahdanau et al. [2015] introduced an attention mechanism between the encoder and decoder to find which source representation to focus on when predicting the current token.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

14.1.2.3.1 The RNN encoder-decoder We first briefly describe the encoder-decoder process proposed by Sutskever et al. [2014] and then introduce the formulation of the solution by Cho et al. [2014]. The main difference being that the former are using two deep LSTMs, one for the encoder and another one for the decoder, while the latter are replacing each LSTM with a RNN having a new type of hidden unit that is much simpler to compute and implement. Given a sequence of inputs (symbol representation) $x = (x_1, ..., x_T)$ and a corresponding sequence of outputs $y = (y_1, ..., y_{T'})$, whose length $T^{'}$ may differ from T, we use the LSTM to estimate the conditional probability $p(y_1, ..., y_{T'} \mid x_1, ..., x_T)$. It is computed in two steps:

1. first obtain the fixed dimensional representation v of the input sequence (it is a summary of the whole input sequence) given by the last hidden state of the LSTM (see Figure (14.2)), and
2. then compute the probability of $y_1, ..., y_{T'}$ with a standard LSTM-LM formulation (the decoder) whose initial hidden state is set to the representation v of $x_1, ..., x_T$.

From Equation (24.1.16), the conditional distribution is given by

$$p(y_1, ..., y_{T'} \mid x_1, ..., x_T) = \prod_{t=1}^{T'} p(y_t \mid v, y_{\leqslant t-1})$$

where $y_{\leqslant t-1} = y_1, ..., y_{t-1}$. Note, the probability distribution $p(y_t \mid v, y_{\leqslant t-1})$ is conditioned on the preceding tokens $y_{\leqslant t-1}$, and on the summary v. Here, each probability distribution $p(y_t \mid v, y_{\leqslant t-1})$ is represented with a Softmax function (see Equation (10.1.4)) over all the words in the vocabulary. Rather than using a standard RNN to compute an output sequence y, given the input sequence x, by iterating the following equation:

$$\begin{aligned} h_t &= \sigma\big(W_{hx}x_t + W_{hh}h_{t-1}\big) \\ y_t &= W_{yh}h_t \end{aligned}$$

the authors used the deep LSTM formulation from Graves (with four layers). They found that it was extremely valuable to reverse the order of the words of the input sentence They required that each sentence ends with a special end-of-sentence symbol "<EOS>", enabling the model to define a distribution over sequences of all possible lengths. This is illustrated in Figure (14.9).

Figure 14.9: The model reads input sentence "ABC" and produces "WXYZ" as the output sentence.

Cho et al. [2014] formulated the solution in a similar fashion. The encoder is an RNN that reads each symbol of an input sequence x sequentially and changes the hidden state of the RNN based on the dynamics (see Equation (12.6.16))

$$h_t = f(h_{t-1}, x_t)$$

where f is a non-linear activation function. After reading the end of the sequence (marked by an end-of-sequence symbol), the hidden state of the RNN is a summary v of the whole input sequence. The decoder is another RNN trained to generate the output sequence by predicting the next symbol y_t given the hidden state h_t. However, both y_t and h_t are also conditioned on y_{t-1} and on the summary v of the input sequence. The formulation is as follows:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- The hidden state of the decoder at time t is given by

$$h_t = f(h_{t-1}, y_{t-1}, v)$$

- The conditional distribution of the next symbol is

$$P(y_t|\, v, y_{\leqslant t-1}) = g(h_t, y_{t-1}, v)$$

  where g is a Softmax function.

This is illustrated in Figure (14.10).

Figure 14.10: RNN Encoder-Decoder with context vector C.

The two components of the RNN Encoder-Decoder are jointly trained to maximise the conditional log-likelihood

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \log P_\theta(y_n|\, x_n)$$

where θ is the set of model parameters and each (x_n, y_n) is a pair of input-output sequence from the training set. We can then use the model to:

1. Generate a target sequence given an input sequence.
2. Score a given pair of input-output sequence, where the score is a probability $P_\theta(y|\, x)$.

The authors also proposed a type of hidden unit (function f), similar to the one in LSTM, but simpler to implement. It includes a reset gate and an update gate adaptively controlling the memory of each hidden unit, while reading or generating a sequence. First, the reset gate r_j is computed by

$$r_j = \sigma\big([W_r x]_j + [U_r h_{t-1}]_j\big)$$

where σ is the logistic sigmoid function, and $[\cdot]_j$ denotes the jth element of a vector. W_r and U_r are weight matrices to be learned. Similarly, the update gate z_j is computed by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$z_j = \sigma\big([W_z x]_j + [U_z h_{t-1}]_j\big)$$

W_z and U_z are weight matrices to be learned. The actual activation of the proposed unit h_j is computed by

$$h_{j,t} = z_j h_{j,t-1} + (1 - z_j)\tilde{h}_{j,t}$$

where

$$\tilde{h}_{j,t} = \phi\big([Wx]_j + [U_r(r \circ h_{t-1})]_j\big)$$

This is illustrated in Figure (14.11).

Figure 14.11: Hidden unit that adaptively remembers and forgets.

When the reset gate is close to 0, the hidden state is forced to ignore the previous hidden state and only reset with the current input, allowing the hidden state to drop any information that is found to be irrelevant later in the future. Since each hidden unit has separate reset and update gates, each hidden unit will learn to capture dependencies over different time scales.

14.1.2.3.2 The RNNsearch Bahdanau et al. [2015] proposed the RNNsearch that consider a bidirectional RNN as an encoder and a decoder that emulates searching through a source sentence during decoding a translation. We use the same notation as in Section (14.1.1) and let the input sequence be $x = (x_1, ..., x_{T_x})$ and denote the context vector z.

A BiRNN consists of forward RNN reading the input sequence as it is ordered from x_1 to x_{T_x}, and a backward RNN reading the sequence in the reverse order from x_{T_x} to x_1. The annotation for each word x_j is obtained by concatenating the forward hidden state $\overrightarrow{h}_j$ and the backward one $\overleftarrow{h}_j$, $h_j = [\overrightarrow{h}_j, \overleftarrow{h}_j]$, so that it contains the summaries of both the preceding words and the following words. Since RNNs have no long-memory, the authors focused on the words around x_j. This sequence of annotations is then used by the decoder and the alignment model to compute the context vector.
In the decoder, each conditional probability in Equation (14.1.1) is given by

$$P(y_i|\, y_1, ..., y_{i-1}; x) = g(y_{i-1}, s_i, z_i) \tag{14.1.8}$$

and each hidden state of the decoder, s_i, is a RNN hidden state computed by

$$s_i = f(s_{i-1}, y_{i-1}, z_i)$$

Note, the probability P is conditioned on a distinct context vector z_i for each target word y_i. Recall, the context vector z_i depends on the sequence of annotations $h = (h_1, ..., h_{T_x})$ to which the encoder maps the input sentence. Each hidden state h_i contains information about the whole input sequence with a strong focus on the parts surrounding the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

i-th word of the input sequence. The context vector z_i is given by Equation (14.1.5). Note, the alignment model, given in Equation (14.1.3), scores how well the inputs around position j and the output at position i match. It is a feedforward neural network jointly trained with all the other components of the system. We can then use the previous translated word y_{i-1}, the hidden state s_i, and the context vector z_i to compute the conditional probability in Equation (14.1.8).

The proposed approach implements a mechanism of attention in the decoder, relieving the encoder from the burden of having to encode all information in the source sentence into a fixed length vector. This is illustrated in Figure (14.12).

Figure 14.12: Alignment attention mechanism generating the word y_t given the input sentence x.

The encoder of the RNNsearch consisted of forward and backward recurrent neural networks (RNN) each having 1000 hidden units. Similarly, the decoder has 1000 hidden units. In both cases, the authors used a multilayer network with a single maxout (see Goodfellow et al. [2013]) hidden layer to compute the conditional probability of each target word.

14.1.2.4 The Transformer

Self-attention is an attention mechanism relating different positions of a single sequence to compute a representation of the sequence. End-to-end memory networks are based on a recurrent attention mechanism.
The transformer, introduced by Vaswani et al. [2017], follows the architecture of the encoder-decoder structure in Section (14.1.2.3). For notation purpose, T is replaced by n, $T^{'}$ by m, and the context vector v by z. A fully connected layers structure is used for both the encoder and decoder.
The encoder is composed of a stack of $N = 6$ identical layers, where each layer has two sub-layers:

1. A multi-head self-attention mechanism.
2. A simple, position-wise fully connected feedforward network.

Around each sub-layer, a residual connection is used, followed by layer normalisation. The output of each sub-layer is

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$f_{ln}\big(x + f_{sub}(x)\big)$$

where $f_{ln}(\cdot)$ is the layer normalisation, and $f_{sub}(\cdot)$ is the function implemented by the sub-layer itself. All sub-layers produces outputs of dimension $d_{model} = 512$.
The decoder is also composed of a stack of $N = 6$ identical layers with sub-layers for each layer. In addition, there is a third sub-layer performing multi-head attention over the output of the encoder stack. Again, around each sub-layer, a residual connection is used, followed by layer normalisation. The self-attention sub-layer is modified to prevent positions from attending to subsequent positions. This is a masking, which ensures that the predictions for position i only depend on the known outputs at positions less than i.

An attention function maps a query and a set of key-value pairs to an output (all being vectors). The output is computed as a weighted sum of the values. The weight assigned to each value is computed by a compatibility function of the query with the corresponding key. There are two attention functions:

1. Scaled dot-product attention: The input are queries and keys of dimension d_k, and values of dimension d_v. The attention function is computed on a set of queries, packed together in a matrix Q. The keys and values are packed into matrices K and V, respectively. The function is given by

$$f_A(Q, K, V) = f_s\big(\frac{QK^\top}{\sqrt{d_k}}\big)V$$

   where f_A is the attention function, f_s is a softmax function (see Equation (10.1.4)).

2. Multi-head attention: The queries, keys, and values are linearly projected h times with different learned linear projections to d_k, d_k and d_v dimensions, respectively. An attention function is performed on each of these projected versions, yielding d_v-dimensional output values. They are concatenated, and projected again, resulting in the final values. These attention functions allow the model to jointly attend to information from different representation subspaces at different positions. In the case of a single attention head, we get

$$\begin{aligned} MultiHead(Q, K, V) &= Concat(head_1, ..., head_h)W^O \\ \text{where } head_i &= f_A\big(QW_i^Q, KW_i^K, VW_i^V\big) \end{aligned}$$

   with projection matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, and $W_i^O \in \mathbb{R}^{hd_v \times d_{model}}$. The authors used $h = 8$ and $d_k = d_v = \frac{d_{model}}{h} = 64$.

Each layer in encoder and decoder also contains a fully connected feedforward network, which is applied to each position separately and identically. It is given by

$$FFN(x) = f_{ReLU}\big(xW_1 + b_1\big)W_2 + b_2$$

where f_{ReLU} is a ReLU activation function. Note, the linear transformations use the same parameters across different positions, but the parameters are different from layer to layer. Equivalently, it is two convolutions with kernel size one. The dimensionality of input and output is $d_{model} = 512$, and the hidden layers have dimensionality $d_{ff} = 2048$. The authors use learned embeddings to convert the input/output tokens to vectors of dimension d_{model}.
For the model to make use of the order of the sequence, the authors inject some information about the relative or absolute position of the tokens in the sequence. They add positional encodings (see Gehring et al. [2017]) with dimension d_{model} to the input embeddings at the bottoms of the encoder/decoder stacks. They use sine and cosine functions of different frequencies.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

14.2 Solving NLP tasks with CNNs

In addition to recurrent neural networks (RNNs) (see Chapter (6)), convolutional neural network (CNNs) (see Section (9.1.1)) have also been used in NLP tasks such as sentiment classification, question-answering, and semantic parsing problems with very promising results. Their success is often contingent on specific architectural and operational choices. Similarly to image processing, they make use of temporal convolution operators or filters applied at multiple resolutions, interspersed with non-linearities and pooling. For example, the convolution operation can be a linear mapping over n-gram vectors obtained by concatenating consecutive word (or character) representations (see Section (1.5.2)). Alternatively, one can perform non-linear interactions within a local context, improving empirical performance in various tasks (see Mitchell et al. [2008], Socher et al. [2013]). Kim [2014] combined CNNs of different filter widths and either static or fine-tuned word vectors. Lei et al. [2015] used a feature mapping operation based on tensor products instead of linear operations on stacked vectors. This way, they directly tap into non-linear interactions between adjacent word feature vectors. In text classification problems, Goldberg [2016] provided a surveys of networks from the perspective of natural language processing research, covering input encoding for natural language tasks, feed-forward networks, convolutional networks, recurrent networks and recursive networks, as well as the computation graph abstraction for automatic gradient computation.

14.2.1 Text classification problems

14.2.1.1 Overview

As discussed in Section (12.6), in NLP there is a need to make predictions on ordered sets of items such as a sequence of words in a sentence. As a result, convolutional neural networks are designed to identify local features in a large pool, and combine them to produce a fixed size vector representations of these features (or context vector), that captures the aspects that are most informative for the prediction task. In that setting, convolutions consist in applying a non-linear filter (a function) on a k-word sliding window over a sequence of words. Applying the filter over that window will create a d-dimensional vector. Thus, the result of the convolution operation will be all the generated d-dimensional vectors. Next, a pooling operation will take the maximum or the average value in each of the d dimensions over the several vectors that resulted from the convolution. The main goal being to focus on the most important features, no matter where they are located, and at the same time producing a fixed-size output.

As an example, we consider the sentence **the quick brown fox jumped over the lazy dog** (see Goldberg [2016]). A sliding window of three words is used in the example and each word is represented by a 2-dimensional embedding vector. The window is represented by the concatenation of the three word embeddings. Each window is then put through a 6×3 filter and the hyperbolic tangent function is applied to each element. A max-pooling operation is then applied, resulting in a final 3-dimensional vector. This is illustrated in Figure (14.13).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 14.13: CNN text classification.

More formally, assuming a word sequence $x = (x_1, ..., x_n)$, each with their corresponding embedding vector v_i^x, $i = 1, .., n$, a convolution layer of width k words works by moving a sliding window of size k over the sequence, creating several instances of windows $w_i = [v_i, v_{i+1}, .., v_{i+(k-1)}]$. A filter, which is usually a linear transformation followed by an activation function, is then applied to each window, resulting in m vectors $p_1, ..., p_m$ with p_i given by

$$p_i = g\big(w_i \cdot W + b\big)\ , i = 1, .., n$$

where g is a non-linear activation function that is applied element-wise, the weight matrix W and vector b are parameters of the network. The m vectors are then passed through a max-pooling layer and a final vector c is obtained. Each element of c is obtained as follows:

$$c_j = \max\big(p_i[j]\big)\ , i = 1, .., n$$

The max-pooling operation results in the most prominent information throughout the various window positions. The idea is that each dimension will capture a particular type of predictors and the $\max$ operation will pick the most relevant one of each dimension and concatenate them together in a final representation vector c with a fixed size.

14.2.1.2 Using tensor products

Since several useful patterns are expressed as non-consecutive phrases, Lei et al. [2015] used a feature mapping operation based on tensor products on stacked vectors. Further, they applied this feature mapping to all possible non-consecutive n-grams in the sequence with an exponentially decaying weight depending on the length of the span.
We let $x \in \mathbb{R}^{L \times d}$ be the input sequence (a document or sentence), where L is the length of the sequence and each $x_i \in \mathbb{R}^d$ is a vector representing the ith word. The consecutive n-gram vector ending at position j is obtained by concatenating the corresponding word vectors

$$v_j = [x_{j-n+1}; x_{j-n+2}; \cdots ; x_j]$$

of size $nd = n \times d$. We let $m \in \mathbb{R}^{nd \times h}$ be the filter matrix being thought of as n smaller filter matrices applied to each x_i in vector v_j. The operator maps each n-gram vector v_j in the input sequence to $m^\top v_j \in \mathbb{R}^h$, where h is the number of feature. Thus, the input sequence x is transformed into a sequence of feature representations

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\left[m^\top v_1, ..., m^\top v_L\right] \in \mathbb{R}^{L \times h}$$

In general, this resulting feature representation is passed through non-linear functions such as the hyper-tangent (elementwise) and aggregated by sum-over or max-pooling operations for later stages of processing. Note, one can replace the basic n-gram vectors and convolution operation with other feature mappings. For example, Lei et al. replaced the three-gram vectors (x_1, x_2, x_3) with the tensor product $x_1 \bigotimes x_2 \bigotimes x_3$, which is an array of coordinate interactions such that each ijk entry of the tensor is given by the product of the corresponding coordinates of the word vectors

$$\left(x_1 \bigotimes x_2 \bigotimes x_3\right)_{ijk} = x_{1i} \cdot x_{2j} \cdot x_{3k}$$

Here, each n-gram in the sequence is expanded into a high-dimensional tensor using tensor products. Hence, the filters are linear mappings over the higher dimensional interaction terms. In the three-gram example, each filter is a three-way tensor with dimensions $d^3 = d \times d \times d$, and the set of f filters, denoted T, is a four-way tensor with dimension $d^3 \times h$, where each d^3 slice of T represents a single filter, and h is the number of filters. That is, the feature dimension. The feature representation $z \in \mathbb{R}^h$ is obtained by multiplying the filter T with the three-gram tensor. The lth coordinate of z is given by

$$\begin{aligned} z_l &= \sum_{ijk} T_{ijkl} \cdot \left(x_1 \bigotimes x_2 \bigotimes x_3\right)_{ijk} \\ & \sum_{ijk} T_{ijkl} \cdot x_{1i} \cdot x_{2j} \cdot x_{3k} \end{aligned}$$

This is equivalent to summing over all the third-order polynomial interaction terms where the tensor T stores the coefficients. However, maintaining the filters as full tensors leads to parametric explosion. As a solution, the authors assumed a low-rank factorisation of the tensor, represented in the Kruskal form. That is, T is decomposed into a sum of h rank-one tensors

$$T = \sum_{i=1}^{h} P_i \bigotimes Q_i \bigotimes R_i \bigotimes O_i$$

where $P, Q, R \in \mathbb{R}^{h \times d}$ and $O \in \mathbb{R}^{h \times h}$ are four smaller parameter matrices and P_i, Q_i, R_i, O_i denote the ith row of these matrices, respectively. Here, it is assumed that the number of rank-one components in the decomposition is equal to the number of feature dimension h. We can rewrite the representation z in vector form as

$$z = O^\top \left(P_{x_1} \bigodot Q_{x_2} \bigodot R_{x_3}\right)$$

where $\bigodot$ is the elementwise product [3]. Thus, in the above example, $z[i, j, k] \in \mathbb{R}^h$ is the feature representation for the three-gram vectors (x_i, x_j, x_k) calculated as

$$z[i, j, k] = O^\top \left(P_{x_i} \bigodot Q_{x_j} \bigodot R_{x_k}\right)$$

Similarly to neural bag-of-words models where the feature representation for a document or sentence is obtained by averaging (or summing) of all the word vectors, the above vectors are aggregated into an h-dimensional feature representation at each position in the sequence. For example, the aggregate representation $z_3[k]$ in the kth position is the weighted sum of all three-gram feature representations ending at position k, given by

$$\begin{aligned} z_3[k] &= \sum_{i<j<k} z[i, j, k] \cdot \lambda^{(k-j-1)+(j-i-1)} \\ & \sum_{i<j<k} z[i, j, k] \cdot \lambda^{k-i-2} \end{aligned}$$

[3] $(a \bigodot b)_k = a_k \times b_k$ for $a, b \in \mathbb{R}^m$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $\lambda \in [0,1)$ is a decay factor putting lower weight to three-grams with longer spans. Note, calculating $z_3[\cdot]$ by enumerating all three-grams require $O(L^3)$ feature mapping operations. To go around, the authors proposed aggregating features via dynamic programming. They considered $z_3[k] = O^\top f_3[k]$, where $f_3[k]$ is a dynamic programming table representing the sum of three-grams feature. That is,

$$f_3[k] = \sum_{i<j<k} \lambda^{k-i-2} \cdot \left(P_{x_i} \bigodot Q_{x_j} \bigodot R_{x_k}\right)$$

Similarly, for the one-gram and two-grams, they got

$$\begin{aligned} f_1[i] &= Px_i \\ f_2[j] &= \sum_{i<j} \lambda^{j-i-1} \cdot \left(P_{x_i} \bigodot Q_{x_j}\right) \end{aligned}$$

The DP tables are calculated recursively

$$\begin{aligned} f_1[i] &= Px_i \\ s_1[i] &= \lambda \cdot s_1[i-1] + f_1[i] \\ f_2[j] &= s_1[j-1] \bigodot Q_{x_j} \\ s_2[j] &= \lambda \cdot s_2[j-1] + f_2[j] \\ f_3[k] &= s_2[k-1] \bigodot R_{x_k} \end{aligned}$$

where $s_1[\cdot]$ and $s_2[\cdot]$ are auxiliary tables. Then, $f_1[i], f_2[j], f_3[k]$ are used to compute $z[k]$. In that setting, the n-gram feature aggregation are performed in $O(Ln)$ matrix addition and multiplication operations. This algorithm maps the input sequence x to a sequence of feature representations $z = z[1:L] \in \mathbb{R}^{L \times h}$ which can be directly aggregated in a classifier, or first transformed and then used as input in another seq2seq feature mapping. The simplest approach is to average the feature representations and apply a softmax output unit on this averaged vector:

$$\begin{aligned} \overline{z} &= \frac{1}{L} \sum_{i=1}^{L} z[i] \\ \tilde{y} &= f_{soft}(W^\top \overline{z}) \end{aligned}$$

where f_{soft} is the softmax function. Here, the authors (1) added a non-linear activation function after each feature representation, $z^{'} = f_{relu}(z+b)$, where f_{relu} is the ReLU function and (2) stacked multiple tensor-based feature mapping layers. The output feature representations of all the layers are averaged within each layer and concatenated. Then, the final prediction is made. This is illustrated in Figure (14.14).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 14.14: CNN architecture using tensor products.

14.3 Transfer learning

14.3.1 Pre-training in NLP

Natural language understanding (NLU) comprises a wide range of diverse tasks such as textual entailment, question answering, semantic similarity assessment, and document classification (see Section (1.5.2)). Although large unlabelled text corpora are abundant, labelled data for learning these specific tasks is scarce, making it challenging for discriminatively trained models to perform adequately. Hence, the ability to learn effectively from raw text is crucial for alleviating the dependence on supervised learning.

Pre-training of NLP models (see Section (1.5.4) is an effective way of improving many language processing tasks such as natural language inference and paraphrasing, entity recognition and question answering. There is a vast literature on pre-training general language representations (see Section (1.5.5)). It acts as a regularisation scheme, enabling better generalisation in deep neural networks (see Erhan et al. [2010]). Nowadays, pre-trained word embeddings are an integral part of modern NLP systems (see Collobert et al. [2008]). However, one does no know apriori the type of optimisation objectives that are most effective at learning text representations which are useful for transfer. Further, there is no consensus on the most effective way of transferring these learned representations to the target task (see Figure (1.9)). Semi-supervised pre-training for NLP is a combination of

1. unsupervised pre-training, and
2. supervised fine-tuning.

Unsupervised pre-training is a special case of semi-supervised learning, where the goal is to find a good initialisation point instead of modifying the supervised learning objective. Most of the proposed methods for pre-training tasks are language modelling (see Bengio et al. [2003]) or its variant.

Among the existing strategies for applying pre-trained language representations to downstream tasks (domain target), we have:

- Feature-based: It leverages pre-training (step [1]) to provide language representations and features to the downstream tasks. It uses task-specific architectures that include the pre-trained representations (learned) as additional features for downstream tasks. For example, Peters et al. [2018] introduced the ELMo that models both

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(1) complex characteristics of word use (such as syntax and semantics), and (2) how these uses vary across linguistic contexts (such as to model polysemy). They extract context-sensitive features from a left-to-right and a right-to-left language model.

- Fine-tuning: It pre-trains a model on language modelling objective (step [1]), and then fine-tune the model on the downstream tasks (step [2]). This is done by introducing minimal task-specific parameters. They require fine-tuning both pre-trained parameters and task-specific parameters on labelled data. For example, Radford et al. [2018] proposed OpenAI GPT, which uses a left-to-right architecture. It is a combination of unsupervised pre-training (step [1]) and supervised fine-tuning (step [2]). They make use of task-aware input transformations during fine-tuning to achieve effective transfer, while requiring minimal changes to the model architecture.

Pre-training and fine-tuning work well when target tasks are of low, or zero, resource in terms of training data, and there is large amount of data available. These pre-training methods obtained very good results in multiple language understanding (NLU) tasks such as sentiment classification, natural language inference, named entity recognition, and question answering (low volume of data).

Both approaches above share the same objective function during pre-training, where they use unidirectional language models to learn general language representations. However, unidirectional language models limits the choice of architectures that could be used for tuning parameters during training. For instance, in token-level tasks such as question answering, it is crucial to incorporate context from both directions. To remedy this problem, Devlin et al. [2018] proposed to transfer knowledge from rich-resource pre-training task to the low/zero-resource downstream tasks. They introduced BERT (Bidirectional Encoder Representations from Transformers) to pre-train deep bidirectional representations from unlabelled text by jointly conditioning on both left and right context in all layers. The approach uses masked language modelling (MLM) pre-training objective and next sentence prediction. The MLM (see Taylor [1953]) randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary ID of the masked word based only on its context. Thus, the model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications. They obtained very good results on eleven natural language processing tasks. Google is leveraging BERT to better understand user searches. This approach is designed for language understanding (NLU), which are usually handled by just one encoder or decoder.

The language generation (NLG) tasks (see Section (1.5.2)) are usually data-hungry, and they are low-resource or even zero-source in terms of training data. An important aspect of the problem is the pre-training and fine-tuning for encoder-decoder based language generation. BERT is designed for language understanding (LU), which are usually handled by just one encoder or decoder. However, language generation (LG) tasks adopt the encoder-decoder based sequence to sequence learning framework. Thus, one can not directly apply BERT like pre-training method on NLG tasks. Some methods have been proposed for pre-training the encoder-decoder model for language generation. For example, XLM (see Lample et al. [2019]) pre-trained BERT-like models both for the encoder and decoder, and achieved the previous state of the art results on unsupervised machine translation. However, the encoder and decoder in XLM are pre-trained separately. Wang et al. [2019c] proposed a seq2aeq pre-training method, called PoDA, which learns representations suitable for text generation task. It jointly pre-trains both the encoder and decoder by denoising the noise corrupted text, and keep unchanged the network architecture in the subsequent fine-tuning stage. Song et al. [2019] introduced MASS (MAsked Sequence to Sequence pre-training) to pre-train NLG tasks. Its encoder takes a sentence with randomly masked fragment (several consecutive tokens) as input, and its decoder tries to predict this masked fragment. Thus, the model can jointly train the encoder and decoder to develop the capability of representation extraction and language modelling. MASS is designed to jointly pre-train the encoder and decoder in two steps:

1. By predicting the fragment of the sentence that is masked on the encoder side, MASS can force the encoder to understand the meaning of the unmasked tokens, in order to predict the masked tokens in the decoder side.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. By masking the input tokens of the decoder that are unmasked in the source side, MASS can force the decoder rely more on the source representation other than the previous tokens in the target side for next token prediction, better facilitating the joint training between encoder and decoder.

14.4 Description of some models

In this section, we present a few popular models that set new state-of-the-art results for various NLP tasks such as question-answering, sentence classification, and sentence-pair regression.

14.4.1 OpenAI GPT

The OpenAI GPT introduced by Radford et al. [2018] is a method to learn a universal representation that transfers with little adaptation to a wide range of tasks. They assume access to a large corpus of unlabelled text and several datasets with manually annotated training examples. The method is a two-stage training procedure:

1. Use a language modelling objective on the unlabelled data to learn the initial parameters of a neural network model.

2. Adapt these parameters to a target task using the corresponding supervised objective.

The model architecture chosen is the Transformer introduced by Vaswani et al. [2017] (see Section (14.1.2.4)), which is used for handling long-term dependencies in text.
In step [1], given an unsupervised corpus of tokens $\mathcal{U} = \{u_1, ..., u_n\}$, the objective is to maximise the likelihood

$$L_1(\mathcal{U}) = \sum_i \log P\big(u_i|\, u_{i-k}, ..., u_{i-1}; \theta\big)$$

where k is the size of the context window, and the conditional probability P is modelled with a NN with parameters θ trained with stochastic gradient descent.
A variant of the Transformer called multi-layer Transformer decoder (see Liu et al. [2018]) is used for the language model. It applies a multi-headed self-attention operation over the input context tokens followed by position-wise feedforward layers to produce an output distribution over target tokens:

$$\begin{aligned} h_0 &= UW_e + W_p \\ h_l &= f_{tb}(h_{l-1})\,, \forall i \in [1, n] \\ P(u) &= f_{sm}(h_n W_e^\top) \end{aligned}$$

where $U = (u_{-k}, ..., u_{-1})$ is the context vector of tokens, n is the number of layers, W_e is the token embedding matrix, W_p is the position embedding matrix, f_{tb} is a transformer block, and f_{sm} is a softmax function.
In step [2], we adapt the parameters to the supervised target task. We are given a labelled dataset $\mathcal{C}$, where each instance consists of a sequence of input tokens $x^1, ..., x^m$, along with a label y. The inputs are passed through the pre-trained model to obtain the final transformer block's activation h_l^m, which is then fed into an added linear output layer with parameters W_y to predict y:

$$P(y|\, x^1, ..., x^m) = f_{sm}(h_l^m W_y)$$

We then get the objective to maximise

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P\big(y|\, x^1, ..., x^m\big)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In addition, the authors found that including language modelling as an auxiliary objective to the fine-tuning helped the learning process. They optimised the objective

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda \cdot L_1(\mathcal{C})$$

with weight λ. Thus, the only extra parameters during fine-tuning are W_y and embeddings for delimiter tokens.
This method is trained on contiguous sequences of text and can directly be applied to some tasks such as classification. However, other tasks have structured inputs such as ordered sentence pairs or triplets of document, so that the method needs to be modified. The authors use a traversal-style approach (transformation), where the structured inputs are converted to an ordered sequence which can be processed by the above method.

14.4.2 BERT

BERT, introduced by Devlin et al. [2018], is a bidirectional encoder representation from transformers using masked language models to enable pre-trained deep bidirectional representations. It is a two step framework:

1. Pre-training: the model is trained on unlabelled data over different pre-training tasks.
2. Fine-tuning: the model is initialised with the pre-trained parameters, and all the parameters are fine-tuned using labelled data from the downstream tasks. Each downstream task has separate fine-tuned models, but are initialised with the same pre-trained parameters.

The architecture is unified across different tasks. The model is a multi-layer bidirectional Transformer encoder based on the model proposed by Vaswani et al. [2017] (see Section (14.1.2.4)). The number of layers (or Transformer blocks) is denoted by L, the hidden size by H, and the number of self-attention heads by A. The model $BERT_{base}(L = 12, H = 768, A = 12, params = 110M)$ is to be compared with the OpenAI GPT, with the difference that BERT uses bidirectional self-attention.
This model set new state-of-the-art results for various NLP tasks such as question-answering, sentence classification, and sentence-pair regression. In the case of question-answer task, the input represent both a single sentence and a pair of sentences. The word sequence references to input token sequence. That is, sentence pairs are packed into a single sequence. In step $[1]$, the pre-training uses two unsupervised tasks:

1. Masked LM (Cloze task): some percentage (about 15%) of the input tokens is randomly masked, which are then predicted. Note, the $[MASK]$ token does not appear during fine-tuning.
2. Next sentence prediction (NSP): pre-train for a binarised next sentence prediction task generated from any monolingual corpus. This is to understand the relationship between two sentences (question-answer).

In step $[2]$: the Transformer allows BERT to model many downstream tasks, which simplifies fine tuning.
In the case of text pairs (such as sentence-pair regression), the input consists of two sentences separated by a special $[SEP]$ token, and it uses the self-attention mechanism to unify the two stages. Note, fine-tuning is more expensive than pre-training.

14.4.3 Masked sequence to sequence pre-training

Following Song et al. [2019], we describe MASS, an unsupervised prediction pre-training framework. We follow the notation from Section (**??**) and consider an unpaired source sentence $x \in \mathcal{X}$. We let $x^{u:v}$ denotes the sentence fragment of x from u to v with $0 < u < v < m$, and m is the number of tokens of sentence x. We further let $x^{\backslash u:v}$ denote a modified version of x, where its fragment from position u to v are masked. We denote $k = v - u + 1$ the number of masked tokens within $[u, v]$. We replace each masked token by a special symbol $[\mathbb{M}]$.
We want to pre-train a sequence to sequence model by predicting the sentence fragment $x^{u:v}$ taking the masked sequence $x^{\backslash u:v}$ as input. The log-likelihood is used as objective function:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}\mathcal{L}(\theta;\mathcal{X}) &= \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}}\log P\big(x^{u:v}\,|\,x^{\backslash u:v};\theta\big)\\ &= \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}}\log\prod_{t=u}^{v}P\big(x_t^{u:v}\,|\,x_{<t}^{u:v},x^{\backslash u:v};\theta\big)\end{aligned}$$

As an example, the authors considered an input sequence has eight tokens with the masked fragment (x_3, x_4, x_5, x_6). See Figure (14.15). The model predicts this masked fragment given (x_3, x_4, x_5) as the decoder input for position 4 to 6 and the decoder takes the mask symbol $[\mathbb{M}]$ as inputs for the other positions. One can choose any neural network for the encoder-decoder framework. The authors considered the Transformer.

Figure 14.15: Masked Seq2Seq framework in MASS

This is to be compared with the GPT framework in Figure (14.16).

Figure 14.16: Masked Seq2Seq framework in GPT

and the BERT framework in Figure (14.17).

Figure 14.17: Masked Seq2Seq framework in BERT

The masked language modelling in BERT and the standard language modelling in GPT are special case of MASS. This can be understood via the hyperparameter k denoting the length of the masked fragment sentence. See Table (14.1).

Contrary to GPT and BERT, MASS is designed to jointly pre-train the encoder and decoder for language generation tasks. By predicting the masked tokens, it forces the encoder to understand the meaning of the unmasked tokens and encourages the decoder to get useful information from the encoder side. Then, by predicting consecutive tokens in the decoder side, the latter can build better language modelling capability than just predicting discrete tokens. At last, by further masking the input tokens of the decoder which are not masked in the encoder, the decoder get more useful information from the encoder side.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Length	Probability	Model
$k = 1$	$P(x^u \mid x^{\setminus u}; \theta)$	masked LM in BERT
$k = m$	$P(x^{1:m} \mid x^{\setminus 1:m}; \theta)$	standard LM in GPT
$k \in (1, m)$	$P(x^{u:v} \mid x^{\setminus u:v}; \theta)$	methods in between

Table 14.1: Masked language modelling.

14.5 Sentence pair modelling

Sentence pair modelling is about modelling the relationship between two sentences, which can be used for paraphrase identification, Natural Language Inference (NLI), Semantic Textual Similarity (STS), Question Answering (QA), among others. We have defined semantic search in Section (1.5.6) as an information retrieval system focusing on the meaning of the sentences rather than the conventional keyword matching. Examples include finding similar sentences between two documents, or a key piece of information in a quarterly earnings report. In this section we will discuss sentence pair modelling, focusing on sentence embeddings and string matching.

14.5.1 Introduction

Sentence processing takes place whenever a reader or listener processes a language utterance [4], either in isolation or in the context of a conversation or a text. Many studies of the human language comprehension process have focused on reading of single utterances (sentences) without context. Extensive research has shown that language comprehension is affected by context preceding a given utterance as well as many other factors. Sentence pair modelling is a fundamental technique underlying many NLP tasks. It measures the degree of equivalence in the underlying semantics of paired snippets of text by identifying whether two sentences express the same meaning, recognising whether a hypothesis can be inferred from a premise, determining how well candidate sentences answer questions, and matching passages with an appropriate question and signaling the text region that contains the answer. Sentence pair modelling is critical for many NLP tasks, such as

- Paraphrase identification (PI): It identifies whether two sentences express the same meaning.
- Natural Language Inference (NLI): This task consists in determining whether an hypothesis sentence can be logically inferred from another premise sentence. It is a classification problem with outcome: entailment, neutral, or contradiction.
- Semantic Textual Similarity (STS): It measures the degree of equivalence in the underlying semantics of paired sentences.
- Question Answering: It can be approximated as ranking candidate answer sentences based on their correctness to the original question.

Several studies have proposed neural network architectures for addressing sentence pair modelling. We distinguish two types of approaches:

1. The sentence encoding approach: It encodes each sentence into a fixed-length vector and then computes sentence similarity directly. In practice, the sentences are separately encoded using a shared encoder and then fed to a classifier network. A model of this type has advantages in the simplicity of the network design and generalisation to other NLP tasks.

[4] Utterance is a spoken word, statement, or vocal sound.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. The sentence pair interaction approach: It takes word alignment and interactions between the sentence pair into account and often show better performance when trained on in-domain data.

In general, neural network models for these tasks rely on pretrained word embedding and compose sentence-level semantics in varied ways. Normally the sentence encoder models are as follows:

- Input Embedding Layer: This layer takes vector representation of words as inputs. Pre-trained word embeddings are commonly used (see GloVe (Pennington et al. [2014])). Some models use subword embeddings (characters or character n-grams).
- Context Encoding Layer: It incorporates word context and sequence order into modelling for better vector representation. This layer often uses CNN (see He et al. [2015b]), recursive neural network like LSTM (Chen et al. [2017]), or highway networks (see Gong et al. [2017]).

In general, we stop at this layer, and directly use the encoded vectors to compute the semantic similarity through the output classification layer. Sentence pair interaction models add to the previous model type:

- Interaction and Attention Layer: It calculates word pair (or n-gram pair) interactions using the outputs of the encoding layer. This is the key component for the interaction-aggregation type of model. Various models put different weights on different interactions, primarily simulating the word alignment between two sentences.
- Output Classification Layer: Present in both sentence encoder models and sentence interaction models, it adapts CNN or MLP to extract semantic-level features on encoder vectors and applies softmax function to predict probability for each class.

It is well known that Subword models are effective in tasks with single sentence input, including language modelling and machine translation. Lan et al. [2018] studied how effective subword-level (character and character n-gram) representations are in sentence pair modelling where the semantic and string similarities between texts matter. They showed that subword models without any pretrained word embedding could achieve new state-of-the-art results on two social media datasets and competitive results on news data for paraphrase identification.
In general, these deep networks require abundant training data, which might be problematic since the construction of labelled data is time-consuming and a labour-intensive process. Several semi-supervised models developed, that take advantage of unlabelled data when only some of the data examples are labelled (see Chapelle et al. [2010]). Choi et al. [2019] proposed to extend a deep probabilistic generative model to make use of unlabelled data. The model is trained to generate a sentence that has a given relationship with a source sentence, with the advantage that both sentences in a pair are utilised together.

14.5.2 Sentence embeddings

Sentence embeddings can be used as features for a logistic regression classifier. They can also be used for transfer learning for other tasks (see Section (14.3)). Sentence embeddings are a well studied area of NLP with a large number of proposed methods. For instance, Kiros et al. [2015] proposed Skip-Thought which trains an encoder-decoder architecture to predict the surrounding sentences. Liu et al. [2016] proposed a deep architecture to model the strong interaction of sentence pair with two coupled-LSTMs. They introduced two coupled ways to model the interdependences of two LSTMs, coupling the local contextualised interactions of two sentences. Then, they aggregated these interactions and use a dynamic pooling to select the most informative features. Conneau et al. [2017] proposed InferSent which uses labelled data of the Stanford NLI dataset and the Multi-Genre NLI dataset to train a Siamese BiLSTM network with max-pooling over the output. Cer et al. [2018] proposed the Universal Sentence Encoder which trains a transformer network and augments unsupervised learning with training on SNLI. Dor et al. [2018] fine-tuned a BiLSTM architecture with triplet loss to derive sentence embeddings for a dataset extracted from Wikipedia. Liu et al. [2018c] proposed Hierarchical Sentence Factorization (HSF), a technique to factorise a sentence into a hierarchical representation, with the components at each different scale reordered into a predicate-argument form. Humeau et al.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2019] addressed the run-time overhead of the cross-encoder from BERT and proposed poly-encoders to compute a score between m context vectors and pre-computed candidate embeddings using attention, but the score function is not symmetric and the computational cost is high for clustering requiring $O(n^2)$ score computations. Reimers et al. [2019] proposed the sentence-BERT (SBERT) which uses siamese network to derive sentence embeddings that can be compared using cosine similarity metric. It is a bi-encoder, and as such, produces a lower performance in comparison with the BERT cross-encoder. To remedy this drawback, Thakur et al. [2020] proposed the Augmented SBERT (AugSBERT), where they used the cross-encoder to label a larger set of input pairs to augment the training data for the bi-encoder. The SBERT bi-encoder is then fine-tuned on this larger augmented training set, getting improved accuracy. Wang et al. [2020] proposed SBERT-WK which embed sentences by dissecting BERT-based word models through geometric analysis of the space spanned by the word representation.
Note, Conneau et al. [2018] proposed a toolkit for evaluating the quality of sentence embeddings when used as features for a logistic regression classifier. The logistic regression classifier is trained on various tasks in a 10-fold cross-validation setup and the prediction accuracy is computed for the test-fold.

14.5.2.1 Sentence encoder models

A typical sentence encoder architecture for solving natural language inference (NLI) problems uses a shared sentence encoder that outputs a representation for the premise and the hypothesis sentences (see Conneau et al. [2017]). Once the sentence vectors are generated, matching methods, via operators $(+, -, \times, /)$, are applied to extract relations between the sentences including concatenation, element-wise product and absolute element-wise difference. The resulting vector, which captures information from both the premise and the hypothesis, is fed into a set of multiple fully-connected layers ultimately passing through a softmax layer. This is illustrated in Figure (14.18).

Figure 14.18: Typical NLI model.

Note, various combinations of the vectors u and v are valid for the concatenation:

$$
\begin{aligned}
&[u, v] \\
&[|u - v|] \\
&[u \circ v] \\
&[|u - v|, u \circ v] \\
&[u, v, u \circ v] \\
&[u, v, |u - v|] \\
&[u, v, |u - v|, u \circ v]
\end{aligned}
$$

For example, Conneau et al. [2017] proposed a bi-directional LSTM (see Section (6.1.5)) with max-pooling as a sentence encoder to create the premise and hypothesis representations. Once the final vector representation is obtained for the hypothesis and premise sentence (the same bi-directional LSTM are used to process both input sentences), in order to produce a combined representation, three matching methods are applied to the two vectors:

1. concatenation of the representation vectors,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. element-wise product,
3. difference between the representation vectors.

Finally, these three operations are all concatenated and the final vector representation for the pair of sentences defined as

$$v = \left[v_p, v_h, |v_p - v_h|, v_p \circ v_h\right]$$

where v_p and v_h are the vector representation for the premise sentence and the hypothesis sentence, respectively. The final vector representation v is then fed to an MLP where a softmax layer is used to make the final classification: entailment, neutral or contradiction.

Nie et al. [2017] proposed a shortcut-stacked encoder for the NLI problem which works by encoding the sentences into fixed length vectors without sharing of information between the premise and the hypothesis sentence. Note, this approach is contrary to pairwise models like the ESIM model (see Chen et al. [2017]) that use cross-features between the sentences. Further, it does not use any attention or memory structure, which have been frequently used in other approaches. The model is composed of multi-layer bi-directional LSTMs, where each layer is fed the output of all the previous layers outputs plus the initial word embeddings, thus creating the mechanism of shortcut connections. The initialisation of the words embeddings is performed with the pre-trained 300D Glove 840B vectors (see Pennington et al. [2014]). More formally, we let f^i_{biLSTM} represents the ith bi-directional LSTM layer, where h^i_t is the output at time t. That is,

$$h^i_t = f^i_{biLSTM}(x^i_t)\,, \forall t \in [1, ..., n]$$

In this model, the first bi-directional LSTM layer just receives as input the word embeddings, while the consequent layers receive the output of all the previous layer plus the initial word embeddings. The input of the ith bidirectional LSTM layer at time t is

$$\begin{aligned} x^1_t &= w_t \\ x^i_t &= \left[w_t, h^{i-1}_t, h^{i-2}_t, ..., x^1_t\right] \end{aligned}$$

where w_t corresponds the word embedding for the word at position t. The results from the final layer of the bi-directional LSTM are processed through a row-max pooling operation, and the final vector representation is obtained. Both representations are combined through the concatenation of the premise and hypothesis, the element-wise product, and the difference between them. For the classification task, the resulting vector from the matching methods is fed into multiple fully connected layers culminating in a softmax layer.

Talman et al. [2018] proposed the hierarchical bi-directional RNN max-pooling as a neural network architecture for NLI. It consists of a stacked bi-directional RNN made of three layers with a max-pooling mechanism in each layer that acts upon the hidden states of the bi-directional RNN layer. Eventually, the results from the three max-pooling mechanisms are concatenated, making up the final string representation. There are shortcut connections between the RNN layers, concatenating the character representations and the output hidden states of the subsequent layer, and feeding them as input to the next RNN layer of the network. This is illustrated in Figure (14.19).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 14.19: Hierarchical bi-directional RNN with max-pooling.

Formally, the inputs of the first, second and third bi-directional RNN layer at instance t are:

$$\begin{aligned} x_t^1 &= e_t \\ x_t^2 &= [e_t, y_t^1] \\ x_t^3 &= [e_t, y_t^2] \end{aligned}$$

where e_t corresponds to the character embeddings, x_t corresponds to the input, and y_t corresponds to the output of the first network layer.

14.5.2.2 Sentence-BERT

We saw in Section (14.4) that BERT set state-of-the-art performance on various sentence classification and sentence-pair regression task. The model relies on a cross-encoder, where two sentences are passed to the transformer network and the target value is predicted. However, it is not suitable for some pair regression tasks such as semantic search, where for a collection of $n = 10,000$ sentences, the pair with the highest similarity requires $\frac{n\cdot(n-1)}{2}$ computations. This is because no independent sentence embeddings are computed in that model. One solution, called BERT embeddings, consists in inputting a single sentence into BERT and then derive fixed-size sentence embeddings, by averaging the BERT output layer (similar to average word embeddings). This is provided by the bert-as-a-service-repository. Reimers et al. [2019] used the siamese network architecture to derive fixed-size vectors from input sentences. A similarity measure, such as cosine or Euclidean distance, is used to find semantically similar sentences. Then, the model is fine-tuned on NLI data.
Extending the BERT model, the SBERT adds a pooling operation to the output of BERT to derive the fixed-size sentence embedding. Three pooling strategies were tried, with the default one being to compute the mean of all output vectors (MEAN strategy). A siamese and triplet networks are created to update the weights to get meaningful sentence embeddings that can be compared with cosine similarity metric. Note, the structure of the model depends on the available data. The following structures were tried

- Classification objective function: Here the sentence embeddings u and v are concatenated with the element-wise difference $|u - v|$ and multiplied with a trainable weight $W_t \in \mathbb{R}^{3n\times k}$:

$$o = f_{SM}\big(W_t(u, v, |u - v|)\big)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where f_{SM} is the softmax activation function, n is the dimension of the sentence embeddings, and k is the number of labels. The cross-entropy loss function is optimised. This is illustrated in Figure (14.20).

- Regression objective function: The cosine similarity metric between two sentence embeddings u and v is computed. The objective function is the mean-squared error loss. This is illustrated in Figure (14.20).
- Triplet objective function: Given an anchor sentence a, a positive sentence p, and a negative sentence n, the triplet loss function tunes the network in such way that the distance between a and p is smaller the distance between a and n. That is, we minimise the loss function

$$\max\big(\|s_a - s_p\| - \|s_a - s_n\| + \epsilon, 0\big)$$

where s_x, $x = a, n, p$, is the sentence embeddings for a, n, p, ϵ is a margin, and $\|\cdot\|$ is the distance metric. Note, ϵ ensures that s_p is at least ϵ closer to s_a than s_n. Here the authors use the Euclidean distance metric and set $\epsilon = 1$.

Figure 14.20: SBERT architecture with classification and regression objective function.

The model is first trained both on the NLI and Multi-Genre NLI dataset, and then fine-tuned with a three-way softmax classifier objective function for one epoch. The numerical settings include a batch-size of 16, Adam optimiser with learning rate $2e-5$, and a linear learning rate warm-up of 10% of the training data. The default pooling strategy is MEAN. The authors replaced the BERT model with the RoBERT one and did not find any significant improvement on experiments. SBERT was shown to be efficient and computationally fast.

14.5.3 String matching

This section extend the models on sentence pair modelling presented in Section (14.5) to the problem of string matching. Santos et al. [2018] considered sentence pair modelling approaches, especially the sentence encoder architecture to tackle the NLI problems, and modified them in order to perform string matching (see Section (3.5.4.2)). This is illustrated in Figure (14.21).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 14.21: Typical string NLP model.

This work was extended by de Sousa Marinho [2018] who experimented with a different stacked RNN encoder, implementing shortcut connections together with stacked RNNs, using self attention mechanisms, hard alignment attention between the strings being compared, or max-pooling over the sequence of outputs of the RNN layers. The proposed neural architectures take as input a pair of strings, or a pair of strings and a common feature, and will output a binary classification on the relation between the inputs as either (1) matching: the two inputs refer to the same real-world entity or object, or (2) non-matching: the two inputs refer to different entities or objects.

14.5.3.1 Mathematical formulation

We let $x \in \mathcal{X}$ be a source sentence, or a document, and $y \in \mathcal{Y}$ be the class label, where $\mathcal{X}$ and $\mathcal{Y}$ are the source and target domain, respectively. We let $x = (x_1, ..., x_n) \in \mathcal{X}^{T_n}$, for some domain $\mathcal{D}$ and time length T_n, be the source sentence with n tokens, and $y = \{1, 0\}$ be the binary class label match and unmatch, respectively. We denote the two sentences of our matching problem as $x^{(1)}$ and $x^{(2)}$. They are converted into strings where the tokens are characters. From a probabilistic approach, we want to find the match or unmatched class label y that maximises the conditional probability of y given a source sentence x, that is, $\arg\max_{y \in \mathcal{Y}} P(y|\, x)$. This means that the classification predicted most often by the classifier is the final classification. Thus, a sentence matching model learns the model parameter θ for estimating the conditional probability

$$P(y|\, x; \theta)$$

Note, we cannot directly predict the class label y from the source sentence x and must first encode the latter in some vectors which are then used as inputs for the classifier. Technically, we map the original input sequence x to a sequence of feature representations z. That sequence can be aggregated to a classifier or one can expose it to non-linear element-wise transformations and use it as an input to another sequence-to-sequence feature mapping. We choose to estimate this conditional probability with an encoder-decoder model, as illustrated in Figure (14.1), where the encoder represents the strings as context vectors z and where the decoder classifies the input vector into a match or unmatched solution y.

In the classifier, we let $\mathcal{Z}$ and $\mathcal{Y}$ be the input and output spaces, $\mathcal{D}$ the probability distribution, D the data set (sample), H the set of hypotheses, f_H the target function, $I(\cdot)$ the indicator function. We let z be an instance or feature vector. For example, we consider the training data set $D = \{(z_1, y_1), ..., (z_m, y_m)\}$ where the instances z_i are independent and identically distributed (i.i.d.) and $y_i = f_T(z_i)$, where f_T is the ground-truth target function. That is, z_i is a vector and y_i takes value in $\{1, 0\}$.
In supervised learning, the learning process aims at constructing a learner f_H minimising the generalisation error. Here, the loss function is given by

$$\mathcal{E}(f_H) = E_Z[I(f_H(z) = f_T(z))]$$

In our setting, we let $\mathcal{Z}$ be the instance space obtained after encoding the source space and $\mathcal{Y}$ be the label space. We define the labelled data set as $L = \{(z_1, y_1), ..., (z_l, y_l)\}$ and consider the binary classification tasks where $\mathcal{Y} = \{1, 0\}$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

14.5.3.2 Base string encoder

In order to encode sentences into fixed-sized representations, the author first considered a recurrent encoder with multiple layers of bi-directional GRU or LSTM units, with dimensionality of 60. Formally, we let $L \in \mathbb{N}$ denotes the number of layers (depth) of the recurrent encoder and N_o the number of nodes for each RNN unit. Then, the recurrent encoder $F(\cdot) : \mathbb{R}^{N_o} \to \mathbb{R}^{N_o}$ is defined as the composition

$$F = F_L \circ \cdots \circ F_1$$

where each component F_l, $l = 1, ..., L$ is a bi-directional LSTM or GRU unit. Here, the authors set $L = 2$. The input is a pair of one-hot binary vectors representing the strings to be compared. The strings are converted into a unicode canonical normalised format and they are padded with a special symbol denoting the beginning and termination of the string. The bi-directional recurrent units take the individual character vector representations as input and work by concatenating the outputs of two GRUs, one processing the vector from left to right and the other from right to left. In the network,

- The first layer generates a sequence of vectors that will be fed as input to the second layer.
- The second layer processes the input and outputs a single vector representation that corresponds to the last output of the bi-directional RNN unit of the second recurrent layer. Note, it can also generate a sequence of outputs, to be summarised later by an attention mechanism and/or a max-pooling mechanism.

Similarly to the Siamese networks for NLI, the parameters of the RNNs are shared across the two parts of the network that process each input string. This encoder is illustrated in Figure (14.22).

Figure 14.22: String encoder with stacked bi-directional GRUs.

In order to improve the baseline string encoder, the author used shortcut connections (see Nie et al. [2017]), which concatenate the character embeddings and the output hidden states of the first layer and feed them as input to the next layer of the network. This encoder is illustrated in Figure (14.23).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 14.23: String encoder with stacked bi-directional GRUs and shortcut connections.

When using shortcut connections, the inputs of the first and second bidirectional RNN layers at time t are given by

$$\begin{aligned} x_t^1 &= e_t \\ x_t^2 &= [e_t, y_t^1] \end{aligned}$$

where e_t corresponds to the character embeddings, x_t corresponds to the input, and y_t corresponds to the output of the first network layer.
The author also considered using an alternative encoder to the bidirectional RNNs called the Variational Bidirectional LSTM (see Shabanian et al. [2017]). Rather than training separately the forward and backward passes, one create a channel of information exchange between the two LSTMs that helps the model to learn better representations. To do so, a variational auto-encoder (VAE) (see Kingma et al. [2014b]) was used that takes as input the hidden states from the two paths of the bidirectional LSTM and maps them to a shared hidden representation of the VAE at each time step. The samples from the VAE hidden state are then used both for reconstructing the LSTM hidden states and feeding forward to the next hidden state.

14.5.3.3 Aggregating RNN states

As discussed in Section (14.1.1), there are several ways for representing the output from the hidden states of the last stacked RNN. For instance, one can directly output the last hidden state of the bi-directional RNN unit (see Sutskever et al. [2014]), or one can leverage the usage of max-pooling over the outputs of the last layer of bidirectional RNNs (see Conneau et al. [2017], Talman et al. [2018]). In the latter, each bi-directional RNN unit in the last layer needs to output the hidden state at each time step. The state variables of the dynamical system in the lth layer are $h_l = \{h_1^l, h_2^l, ..., h_n^l\}$, and the output of the last bi-directional RNN unit is the hidden states $h_L = \{h_1^L, h_2^L, ..., h_n^L\}$. Since $h_i^l \in \mathbb{R}^d$, then h_l is a matrix of size (n, d). Applying row max-pooling to the matrix h_L will result in a vector representation containing the maximum value of each row of that matrix. This encoder with max-pooling is illustrated in Figure (14.24).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 14.24: String encoder with stacked bi-directional GRUs and max-pooling.

Another way of obtaining the representation of a string from the hidden states of the last bidirectional RNN, F_L, is to consider self-attention mechanisms. Following Raffel et al. [2016], we consider a simplified attention mechanism over the hidden states of the last bidirectional RNN layer, F_L, of the network. At each time step i, the simplified attention is formulated as follows:

$$e_i = A_{net}(h_i^L)\,,\, \alpha_i = \frac{e^{e_i}}{\sum_{k=1}^{T_x} e_k}\,,\, z = \sum_{i=1}^{T_x} \alpha_i h_i^L \tag{14.5.9}$$

where $A_{net}(\cdot; W)$ is a learnable function with weight matrix W, which only depends on the hidden state h_i at time step i. In general, the function A_{net} is set to $\tanh$. In practice, the output of the last bi-directional RNN unit, the hidden states $h_L = \{h_1^L, ..., h_n^L\}$, is fed to the non-linear function A_{net}, which becomes the input of a softmax function yielding a weight vector. The vector α represents the weights that are used to calculate the context vector z as a weighted sum of the states h_L. Recall, $h_i^l \in \mathbb{R}^d$, so that, in the product $\alpha_i h_i^L$, all the dimensions of h_i^L are multiplied by the scalar α_i. This simplified attention is illustrated in Figure (14.7).

Figure 14.25: String encoder with stacked bi-directional GRUs and self attention mechanism.

The author also tested a hard alignment attention mechanism that starts by taking the two representations produced by the sentence encoder, $h_L^{(1)}$ and $h_L^{(2)}$, and then compute an elementwise dot product between them, resulting in a matrix M. After this, a softmax is applied to both the rows and columns of M, resulting in two weight matrices denoted W_1 and W_2. They are combined with the first and second representations of the input strings, $h_L^{(1)}$ and $h_L^{(2)}$, using a dot product, resulting in two matrices of weighed inputs inW_1 and inW_2. Note, this approach is different calculating the weights and calculating a context vector as a weighed sum of the hidden states in a self-attention mechanism. Finally,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

each matrix column of the weighed inputs is concatenated to its corresponding input vector representation, resulting in two new representations for each string. This mechanism is illustrated in Figure (14.26).

Figure 14.26: String encoder with stacked bi-directional GRUs and hard alignment attention mechanism.

Note, these new representations for each input are matrices, and they need to be transformed into fixed-length vectors in order to obtain a final representation. To do so, one can apply a self-attention layer, a max-pooling mechanism, or feeding the representation to another layer of bi-directional RNNs that only outputs the result from the last time step.

14.5.3.4 Hierarchical BiRNN with max-pooling

The author also considered to adapt the hierarchical bi-directional RNN max-pooling, which is a neural network architecture proposed by Talman et al. [2018] for NLI to the problem of string matching. This is illustrated in Figure (14.19). Here, the RNN chosen for the encoder was the LSTM, but the author also suggested using a GRU.

14.5.3.5 Using common features

The author proposed to use common features to both strings being compared. They must be represented and combined with both string representations. This was done by concatenating the features representation to the existing combined representation of both strings. This illustrated in Figure (14.27).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 14.27: String encoder with common features.

14.5.3.6 Architecture for toponym matching

The architecture for toponym matching is the one proposed by Santos et al. [2018] and described in Section (3.5.4.2). The results from each matching method are concatenated, resulting in the final representation that is going to be used for the classification. Here, the author considered two approaches:

1. A simple feed-forward layers processing the combined representations (as in Santos et al. (see Figure (3.28))): The combined final representation is passed as input to feed-forward network layers. The first layer uses a simple combination of the input together with a ReLU activation function. This is followed by another layer that uses a sigmoid activation function to produce the final output resulting in a probability value that can be thresholded at 0.5 to either the value matching or nonmatching.

2. A highway network: It is a neural architecture enabling the training of extremely deep networks using simple stochastic gradient descent (see Srivastava et al. [2015b]). A feed-forward network of L layers only applies a non-linear transform H at each layer on its input x to produce an output y. That is, $y = H(x \cdot W_H)$. A highway network, additionally defines two non-linear transformations T and C such that

$$y = H(x \cdot W_H) \circ T(x \cdot W_T) + x \cdot C(x \cdot W_C)$$

   where T is referred to as the transform gate, and C as the carry gate, since they express how much of the output is produced by transforming the input and carrying it, respectively. Setting $C = 1 - T$, a highway layer can smoothly vary its behaviour between that of a plain layer and that of a layer which simply passes its inputs through. See details in Section (9.2.1).

14.5.4 Semantic search

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Part IV

Stochastic Control

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 15

Markov Decision Problems

15.1 Presentation

15.1.1 Decision-making problems

Markov decision problems (MDPs) combine evaluative feedback (as in bandits) with an associative aspect, by choosing different actions in different situations. They are sequential decision-making problems where decisions (via action or control) made in each state affect the trajectory of the states visited by the system over a particular time. In turn, the trajectory affects the performance of the system measured by a specific metric (total discounted reward). Thus, MDPs involve delayed reward and the need to trade off immediate and delayed rewards.
Traditionally solved with dynamic programming (DP), these problems require to know the transition probabilities, which are difficult to derive for large-scale problems. The use of simulation for solving MDPs became popular because it can bypass the transition probability model for solving large-scale problems with finite action space. The simulation-based approach for solving MDPs, rooted in the Bellman equations (see Bellman [1954]), is closely tied to solving control problems in discrete-event dynamic systems. These methods are called Value Iteration (see Bellman [1957]) and Policy Iteration (see Howard [1960]).

In the case of large complex systems, it is easier to produce simulator than developing exact mathematical models to devise the transition probabilities. This is the case for simulation-based approaches which have been given different names such as reinforcement learning (RL), neuro-DP, and approximate or adaptive DP (ADP). In general, RL and ADP are divided into at least two branches:

1. System running on-line: it uses algorithms on a real-time basis within the system. It is associated to RL and works with Q-function which avoids transition probabilities.

2. System running off-line: it solves large-scale MDPs where the transitions probabilities are available. It is associated to ADP and works with value function of DP.

However, this distinction between systems running on-line or off-line is not always clear (see Section (1.2.3.2)). Note, reinforcement learning agents can deal with a wide range of non-Markovian environments by having a memory of their past (see Lin [1993]).

See Kaelbling et al. [1996] for a survey on reinforcement learning, lectures from Li et al. [2017], Sompolinsky [2017], Levine et al. [2017], Simonini [2018], and textbooks by Szepesvari [2010], Mohri et al. [2012], Sutton et al. [2017], Gosavi [2014], among others.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

15.1.2 Description

Reinforcement learning (RL) consists in solving problems involving an agent (or decision-maker) interacting with an environment, which provides numeric reward signals. The objective being to construct a control policy maximising the agent's performance. A natural measure of performance is the discounted cumulative reward $\sum_{t\geqslant 0} r_t$, where r_t is the reinforcement received after the transition from time t to $t+1$. Note, the reinforcement signal can be positive (reward), negative (punishment), or null. Hence, the goal is to learn how to take actions in order to maximise reward. Some important terms regarding reinforcement learning are:

- Action a: All the possible moves that the agent can take.
- State s: Current situation returned by the environment.
- Reward r: An immediate return send back from the environment to evaluate the last action. It depends on the state s and the action a.
- Policy π: The strategy that the agent employs to determine next action based on the current state of the environment. It can be deterministic or stochastic.
- Value: The expected long-term return with discount, as opposed to the short-term reward.

The agent is required to select an action from a set of actions in a subset of states. Those states are called decision-making states. We let $\mathcal{S}$ be the set of states and $\mathcal{A}$ is the set of actions. Formally, we have:

Definition 15.1.1 *Let $d, m \in \mathbb{N}$ fixed. A state space $\mathcal{S}$ (resp. an action space $\mathcal{A}$) is a Borel subset of $\mathbb{R}^d$ (resp. $\mathbb{R}^m$). We call $\mathcal{B}(\mathcal{S})$ (resp. $\mathcal{B}(\mathcal{A})$) its Borel σ-field.*

As a result of selecting an action, the system transitions to another state in a probabilistic manner. That is, the probability that the process moves into its new state $s^{'}$ is influenced by the chosen action. It is given by the state transition function. Thus, the next state $s^{'}$ depends on the current state s and the action a. These actions are stored in a policy. In a deterministic action, the agent can select a fixed action in each state, while in a stochastic policy each action is chosen with some fixed probability. In a stationary policy, the action chosen in a state does not change with time.

Statement 15.1.1 *The sequence of events is as follows:*

$$s_t \rightarrow a_t \rightarrow (s_{t+1}, r_{t+1})$$

The actions change the state of the environment, which affects future movement possibilities. This is illustrated in Figure (15.1).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 15.1: Agent-environment interaction

A few terms are often used:

- Experience: It is a quadruple $(s_t, a_t, s_{t+1}, r_{t+1})$ meaning that the execution of an action a_t in a state s_t results in a new state s_{t+1} and reinforcement r_{t+1}.
- Lesson: It is a temporal sequence of experiences starting from an initial state to a final state, where the goal may or may not be achieved.
- Episode: When the return is a finite sum of rewards

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T$$

  where T is a final time step. In that case, the agent-environment interaction breaks naturally into subsequences called episodes. Each episode ends in a terminal state followed by a reset to a starting state. Tasks with episodes are called episodic tasks. The termination time T is a random variable varying from episode to episode. When the agent-environment interaction does not break, it is called a continuing task.

In traditional RL, we assume that there is a teacher providing the learner with reinforcement signals. Hence, giving an agent a reinforcement mechanism signify giving him a goal. To do so, we need to design a reinforcement function specifying the goal of the task. An example of such a mechanism would be a neural network model (see Chapter (5)), where the values are approximately:

- $s \sim X$ the input
- $a \sim Y$ the output

and the policy π reflects the calculation performed by the network to generate Y from X. In that sense, it is a function from $\mathcal{S}$ to $\mathcal{A}$, that is, $\pi : \mathcal{S} \to \mathcal{A}$ (here it is a deterministic function, see Definition (15.3.3)). Thus, given some data, we can use a network to learn the policy π. Further, the reward is a function from $\mathcal{S} \times \mathcal{A}$ to $\mathbb{R}$, that is, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. At time step t and state s_t, the agent chooses an action a_t according to its policy $\pi(a_t|s_t)$, then the state and the environment changes to new state s_{t+1} according to the dynamics

$$P(s_{t+1} = s^{'}, r_{t+1} = r^{'}|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0)$$

which depends on the entire history of the system. In a Markov Decision Process (MDP), the system only depends on the previous state and action. The dynamics is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$P(s_{t+1} = s^{'}, r_{t+1} = r^{'}|s_t, a_t)$$

and the agent receives the reward $r_t = r(s_t, a_t)$. The process continues.
The goal of reinforcement learning is to find the optimal policy π^* maximising the expected cumulative reward (see Appendix (23.2.1.1))

$$\pi^* = \arg\max_{\pi} E[\sum_{t\geqslant 0} r_t|\pi]$$

with $s_0 \sim p(s_0)$, $a_t \sim \pi(\cdot|s_t)$, $s_{t+1} \sim p(\cdot|s_t, a_t)$. The RL anatomy is

$$\text{generate samples} \implies \text{fit a model / estimate return} \implies \text{improve the policy} \hookleftarrow$$

Similarly to supervised learning (SL), which is guided by error gradient signals, reinforcement learning is guided by feedback from the environment in the form of reward signals. However, some of the main differences are:

- Reward versus desired labels: There are no target labels in RL. Feedback is given in terms of some measures of success or failure, but the correct response is usually not provided. Since the system must work with partial information, we can not quantify how far we are from the right answer and as such there is no error gradient to compute.
- Dynamic versus static interaction with environment: SL (ANN) assumes a fixed stationary distribution of examples provided by the environment. However, in some real life problems the environment is dynamic in the sense that the next examples provided by the environment depends on the previous behaviour of the agent. Thus, we need to learn the desired trajectories rather than a fixed input output function. Since the statistics of the reward changes with time (it is path dependent) the actual reward may be delivered only later in the future, after a long sequence of actions.
- Look up tables versus distributed representations: Due to the inherent complexity of the task of the agent, the most common implementation of RL is in the form of a look-up table representations. It guarantees convergence of the algorithms for low dimensional problems. In the case of high dimension, implicit distributed representations such as neural networks are still under investigation.
- Markov decision process (MDP): Presently RL is performed in the theoretical framework of MDP. However, the underlying Markovian assumption is an important restriction, leaving out many real life situations.

Some examples of RL problems are:

- Robot locomotion: Objective: make the robot move forward; State: angle and position of the joints; Action: torques applied on the joints; Reward: 1 at each time step upright plus forward movement.
- Video games: Objective: complete the game with the highest score; State: raw pixel inputs of the game state; Action: game control (left, right, up, down); Reward: score increase/decrease at each time step.
- Go game: Objective: win the game; State: position of all pieces; Action: where to put the next piece down; Reward: 1 if win at the end of the game, 0 otherwise.
- Optimal portfolio selection: Objective: optimise the wealth of an agent at some future time; State: stock price; Action: choose between two different investments, a risky asset and a risk-free one; Reward: payoff at each time step. See details in Appendix (28.3).
- Option pricing: it can be cast into an optimal portfolio selection problem (see Section (22.1)).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

15.2 Learning with stationary reward

A Markov process is a memoryless random process with the Markov property. It is a tuple $(\mathcal{S}, \mathbb{P})$ where $\mathcal{S}$ is a finite set of states and $\mathbb{P}$ is a state transition probability matrix, $\mathbb{P}_{ss'} = P(s_{t+1} = s'|s_t = s)$.
A Markov reward process is a stochastic process which extends either a Markov chain or continuous-time Markov chain by adding a reward rate to each state (see Howard [1971]). An additional variable records the reward accumulated up to the current time. Features of interest in the model include expected reward at a given time and expected time to accumulate a given reward.
We consider a Markov reward process (MRP) defined by the tuple $(\mathcal{S}, \mathcal{R}, \mathbb{P}, \gamma)$ where $\mathcal{S}$ is a finite set of possible states, $\mathcal{R}$ is the distribution of reward given $(state, action)$ pair, $\mathbb{P}$ is the transition probability, $\gamma \in [0, 1]$ is the discount factor. The reward function is given by $\mathcal{R}_s = E[r_{t+1}|s_t = s]$.

15.2.1 Description

To account for the lack of explicit gradient signals (no target labels) one solution is to use neural networks (NNs) with built-in stochasticity. This noise will be used to form exploration of the state space of $\mathcal{S}$ such that an estimate of the gradient is implicitly evaluated, in a process similar to the stochastic gradient descent discussed in the context of online learning.
In Section (10.1.1), we have considered a simple two-layer network with input layer X, weights W and output Y. So far, we have assumed fixed parameters W_i and some input X_i. However, in reinforcement learning, after performing an action, the system receives a reward. It then updates its weights accordingly, with the desire to maximise future rewards. Thus, we let the policy be represented by a network where the input represent the state, the output is the action selection probabilities, and the weights are the policy parameters. We let W be the vector of parameters and ρ is the performance of the corresponding policy, that is, the average reward per step.
At each time step we do:

1. The environment provides an input X_{input} to the network according to some probability distribution $P(X_{input})$.
2. The network use the transition probabilities to generate the outputs $\{Y_i\}$ (can be the activities of all neurons in the network) in a feedforward manner corresponding to the action taken by the system.
3. Based on the output, the environment creates the reward signal which is a scalar r. The environment is generally stochastic, so that there is a probability distribution for the reward given the outputs, that is, $P(r|\{Y_i\})$.
4. The reward signal is a global scalar broadcasted to the entire network. All connections W_{ij} are updated in a way that will maximise the average reward. That is, maximising $\rho =< r >= R(W)$ [1], where the average is with respect to
   (a) The inputs X_{input}, according to $P(X_{input})$.
   (b) The conditional distribution of the outputs Y_i, according to $g_i(y, W_i, X_i)$.
   (c) The conditional rewards distribution r_i, denoted $P(r|\{Y_i\}, X_{input})$.

It is generally assumed that all of these external distributions are stationary, that is, they are independent of time. The goal is to find

$$\arg\max_{W} < r >$$

over all the parameters of the network. We want to find the set of weights that guarantees the highest possible rewards, over enough time. One way of achieving such a result is to use the REINFORCE algorithm.

[1] Given two random variables X, Y in L_2, the inner product is $< X, Y >= E[XY]$. We denote $< X >= E[X]$ the expectation of X. In the case where $X = (X_1, ..., X_n)$ we can write $< a, X >= \sum_{i=1}^{n} a_j X_j$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

15.2.2 The REINFORCE algorithm

In this algorithm (see Williams [1992]) (Derivation is given in Section (17.1.4)) the weights are updated according to the following rule:

$$\Delta W_{ij} = \alpha_{ij}(r - b_{ij})e_{ij} \ , \ e_{ij} = \frac{\partial \ln g_i}{\partial W_{ij}}$$

where α_{ij} is the learning rate and g_i is the the conditional probability mass function given in Equation (10.1.3). This value is independent of the reward and the output. r is the instantaneous global reward signal. b_{ij} is the reward threshold, also known as the baseline, and it is independent from the reward and the output. e_{ij} is the eligibility trace representing the responsibility of a particular connection to the generation of the output Y_i, and is therefore a stochastic quantity depending on the input and the network activities, but not on the reward.
The claim of the algorithm states: for a given vector W, we have

$$\sum_{i,j} < \Delta W_{ij} > \frac{\partial R(W)}{\partial W_{ij}} > 0$$

where $< r >= R(W)$ is a function of W. That is, on average, the change in weights has a positive projection on the reward gradient.
One can prove that if $\alpha_{ij} = \alpha$ (a constant) then

$$< \Delta W_{ij} > \propto \frac{\partial < r >}{\partial W_{ij}}$$

implying that we are moving in the exact direction of the gradient.
Proof: Given the policy $g_i = P(Y_i = y|W_i, X_i)$ (see Equation (10.1.3)), from the normalisation of the mass function (see Lemma (24.1.2) for the pmf), we have

$$\sum_{y_i} g_i(y_i, W_i, X_{input}) = 1$$

From the chain rule, and since r is independent from W, we have

$$\frac{\partial \ln g_i}{\partial W_{ij}} = \frac{1}{g_i(y_i, W_i, X_i)} \frac{\partial g_i}{\partial W_{ij}}$$

The eligibility trace is a random variable with mean zero (over y_i),

$$< e_{ij} >=< \frac{1}{P(Y_i = y_i|W_i, X_i)} \frac{\partial g_i}{\partial W_{ij}} > \sum_{y_i} \frac{\partial g_i}{\partial W_{ij}} = 0$$

From Theorem (24.1.4), we have

$$\begin{aligned}
< \Delta W_{ij} > &= \sum_{r, X_{input}, y_i} \alpha_{ij}(r - b_{ij}) \frac{1}{g_i(y_i, W_i, X_i)} \frac{\partial g_i}{\partial W_{ij}} \left[P(X_{input}) P(Y_i = y_i|W_i, X_i) P(r|\{Y_i\}, X_{input}) \right] \\
&= \alpha_{ij} \sum_{r, X_{input}, y_i} P(X_{input}) P(r|\{Y_i\}, X_{input}) r \frac{\partial g_i}{\partial W_{ij}} - \alpha_{ij} \sum_{X_{input}} P(X_{input}) b_{ij} \frac{\partial g_i}{\partial W_{ij}} \\
&= \alpha_{ij} \frac{\partial}{\partial W_{ij}} \sum_{r, X_{input}, y_i} P(X_{input}) P(r|\{Y_i\}, X_{input}) g_i(y_i, W_i, X_{input}) \\
&= \alpha_{ij} \frac{\partial}{\partial W_{ij}} R(W)
\end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where we have used the fact that $P(r|\{Y_i\}, X_{input})$ is independent of W to pull the gradient with respect to W outside of the sums.
The result is

$$\sum_{i,j} < \Delta W_{ij} > \frac{\partial R(W)}{\partial W_{ij}} = \sum_{i,j} \alpha_{ij} \left(\frac{\partial R(W)}{\partial W_{ij}}\right)^2$$

which proves the claim.
Thus, for $\alpha_{ij} = \alpha$ the algorithm is a stochastic ascent process. For a given weight W, we get

$$\begin{aligned} \Delta W &= < \Delta W > + \eta \\ &= \alpha \nabla_W R(W) + \eta \end{aligned}$$

so that η is a zero mean noise ($< \eta >= 0$). As a result, all the theorems regarding convergence of STA theory apply to this algorithm.

15.2.2.1 Some examples

15.2.2.1.1 Bernoulli neurons In the case of Bernoulli elements we have $g_i(Y_i = 1) = P_i$ and $g_i(Y_i = 0) = 1 - P_i$. Assuming a neuron with no input (a neuron whose input is always 1), we want to learn the value of P_i maximising the reward (Two arm bandit problem). We can compute the eligibility trace as

$$\begin{aligned} e_{ij} &= \frac{\partial \log g_i}{\partial P_i} \\ &= \frac{1}{g_i} \frac{\partial g_i}{\partial P_i} \\ &= \frac{1}{P_i(1 - P_i)} (Y_i - P_i) \end{aligned}$$

which is dependent on the value of Y_i. We can also compute its expectation as

$$\begin{aligned} < e_{ij} > &= \frac{1}{P_i(1 - P_i)} < Y_i - P_i > \\ &= \frac{1}{P_i(1 - P_i)} < Y_i - < Y_i >> \\ &= \frac{1}{< \delta Y_i^2 >} < Y_i - < Y_i >> \end{aligned}$$

Thus, the expected value of the eligibility trace is zero and its variance is $< \delta Y_i^2 >$.
If we choose $\alpha_{ij} = \eta_i P_i(1 - P_i)$ and $b_i = 0$, the learning rule simplifies to

$$\Delta P_i = \eta_i r(Y_i - P_i)$$

Assuming $r > 0$, it will converge to $P_i = 1$ if $< r|1 >< r|0 >$ and vice versa.

15.2.2.1.2 Binary LN neurons In the case of a binary LN neuron we have

$$\begin{aligned} P_i &= g_i(1, W_i, X_i) \\ &= \frac{1}{1 + e^{-W_i \cdot X_i}} \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

so that P_i is a function $h_i \equiv W_i \cdot X_i$. We get

$$\begin{aligned}\frac{\partial P_i}{\partial W_{ij}} &= \frac{X_{ij}e^{-W_i \cdot X_i}}{\left(1+e^{-W_i \cdot X_i}\right)^2} \\ &= P_i(1-P_i)X_{ij}\end{aligned}$$

Since we are varying W, we need to calculate the eligibility trace

$$\begin{aligned}e_{ij} &= \frac{\partial \log g_i}{\partial P_i} \cdot \frac{\partial P_i}{\partial W_{ij}} \\ &= \frac{1}{P_i(1-P_i)}(Y_i - P_i)P_i(1-P_i)X_{ij} \\ &= (Y_i - P_i)X_{ij}\end{aligned}$$

If we choose $\alpha_{ij} = \eta > 0$ and $b_{ij} = 0$, the earning rate simplifies to

$$\Delta W_{ij} = \eta r(Y_i - P_i)X_{ij}$$

The form $(Y_i - P_i)X_{ij}$ is the classic Hebbian unsupervised learning.
To guarantee convergence we need some bounds on the fluctuation of the reward, which can be sensitive to the choice of the baseline b_{ij}. It can make the convergence of the learning algorithm very slow so that one we be careful when choosing the baseline. A popular choice is

$$b_{ij} = \overline{r}(t)$$

where

$$\overline{r}(t) = r(t-1)(1-\gamma) + \gamma\overline{r}(t-1)\,,\, 0 < \gamma < 1$$

It is a sort of integration of all of the rewards up to the current point in time, where the rewards from the past diminish exponentially. Thus b_{ij} is an estimate of the expected reward based on past history. It is a temporary average of the reward where γ is a decay parameter controlling how far back in time the system looks. It diminishes the rewards from the far past to nothing, while keeping the rewards from the near past relevant.

15.3 Markov decision process

As discussed in Section (15.1), in reinforcement learning (RL), the goal of the agent is to choose some actions in each state that optimises the value of some performance metric. We want to find a function π (policy) that specifies the action, $a = \pi(s)$, that the agent will choose when in state s. For instance, the agent can choose a policy that will maximise the average reward, or the discounted reward.
Markov decision processes (MDP) (see Puterman [1994], Ross [1996], Bertsekas [2005]) are discrete time stochastic control processes providing a mathematical framework for modelling decision making in situations where outcomes are partly random and partly under the control of a decision maker. Thus, MDPs can be used to formulate the RL problem. An MDP is a Markov reward process (MRP) with decision where $\mathcal{A}$ is a finite set of actions. The current state completely characterise the state of the world. The MRP is defined by the tuple $(\mathcal{S}, \mathcal{R}, \mathbb{P}, \gamma)$, while the MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$, where $\mathcal{S}$ is the set of possible states, $\mathcal{A}$ is the set of possible actions, $\mathcal{R}$ is the distribution of reward given $(state, action)$ pair, $\mathbb{P}$ is the transition probability, $\gamma \in [0,1]$ is the discount factor. The state transition matrix is $\mathbb{P}_{ss',a} = P(s_{t+1} = s'|s_t = s, a_t = a)$ and the reward function is $\mathcal{R}_{s,a} = P[r_{t+1} = r'|s_t = s, a_t = a]$. The

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

single-stage expected reward is $R(s,a) = E[r_{t+1} = r^{'}|s_t = s, a_t = a]$.
In the Markov decision process (MDP), the decision process is given in Algorithm (30).

Algorithm 30 Markov Decision Algorithm

Require: : training set S, number of training steps T
1: Initialise the state $s_0 \sim p(s_0)$
2: **for** $t = 0$ to $T - 1$ **do**
3: Agent select action a_t
4: Environment samples reward $r_t \sim \mathcal{R}(\cdot|s_t, a_t)$
5: Environment samples next state $s_{t+1} \sim \mathbb{P}(\cdot|s_t, a_t)$
6: Agent receives reward r_t and next state s_{t+1}
7: **end for**

There exists several types of MDP problems (see Ross [1996]) which are classified as follows:

1. Discrete time
   - countably finite state and action spaces
   - countably infinite state and/or action spaces
2. Continuous time
   - requires partial differential equations, the Hamilton-Jacobi-Bellman (HJB) equation
   - limiting case of Bellman equation as time-step tends to zero.

In the case where the system is a stochastic process, the problem becomes that of a stochastic control problem. The continuous time MDP is presented in Section (15.4). We will then present in Chapter (17) some problems of interest to reinforcement learning associated to the case of countably finite state and action spaces. We will also give some hints to infinite action space.

15.3.1 The problem

15.3.1.1 Description

As discussed above, a Markov decision processes (MDP) can be seen as an extension of Markov chains with the addition of actions (allowing choice) and rewards (giving motivation). The main goal is to find a policy that specifies the action that the agent will choose when in a given state. That is, at each time step, the process is in some state s, and the agent may choose any action a that is available in state s. As a result of selecting an action, the system transitions to another state in a probabilistic manner. The probability that the process moves into its new state $s^{'}$ is influenced by the chosen action. It is given by the state transition function.
In general, the dynamics of the environment depends on the entire history of the system (all the previous states and actions). In that case, the transition probability is given by

$$P\big(s_{t+1} = s^{'}, r_{t+1} = r^{'}|(s_t, a_t), (s_{t-1}, a_{t-1}), ..., (s_0, a_0)\big)$$

In the case of the Markov decision process (MDP), the current change in state and the reinforcement received only depend on the previous state and action. This is the Markov property (see Appendix (24.6.3)). A Markov chain is a stochastic process with the Markov property (see Definition (24.6.1)) which is written as

$$P\big(s_{t+1} = s^{'}, r_{t+1} = r^{'}|(s_t, a_t), (s_{t-1}, a_{t-1}), ..., (s_0, a_0)\big) = P\big(s_{t+1} = s^{'}, r_{t+1} = r^{'}|(s_t, a_t)\big)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

if both conditional probabilities are well defined. In an MDP, the probabilities given by P completely characterise the dynamics of the environment. That is, the probability for each possible value for s_{t+1} and r_{t+1} depends on the immediately preceding state and action s_t and r_t, respectively. As discussed in Remark (24.6.2), Markov chains can be described by a sequence of directed graphs, where the edges of graph t are labelled by the probabilities of going from one state at time t to the other states at time $t+1$, denoted $P(X_{t+1} = x|X_t = x_t)$. The latter is the probability of the state of the machine, which describes the statistical behaviour of the machine with an element x_t of the state space as input.

We assume that the state s is chosen from a finite set $\mathcal{S}$ and, similarly, the actions a is chosen from a finite set $\mathcal{A}$. We let $S = (s_t)_{t\in\mathbb{N}}$ be a sequence of $\mathcal{S}$-valued random variables, $(\mathcal{F}_t)_{t\in\mathbb{N}}$ be the canonical filtration generated by S, and $A = (a_t)_{t\in\mathcal{N}}$ be an $(\mathcal{F}_t)_{t\in\mathbb{N}}$-adapted sequence of $\mathcal{A}$-valued random variables. We define a Markov decision process as follows:

Definition 15.3.1 *The process (S, A) is said to be a Markov decision process if and only if there exists two functions*

$$p : \begin{cases} \mathcal{B}(\mathcal{S}) \times \mathcal{S} \times \mathcal{A} \to [0,1] \\ (B, s, a) \to p(B|\, s, a) \end{cases} \quad \textit{and } \pi : \begin{cases} \mathcal{B}(\mathcal{A}) \times \mathcal{S} \to [0,1] \\ (B^{'}, s) \to \pi(B^{'}|\, s) \end{cases}$$

such that for every $(t, B, B^{'}) \in \mathbb{N} \times \mathcal{B}(\mathcal{S}) \times \mathcal{B}(\mathcal{A})$

$$\begin{cases} P(a_t \in B^{'}|\, \mathcal{F}_t) = \pi(B^{'}|\, s_t) \\ P(s_{t+1} \in B|\, \mathcal{F}_t) = P(B|\, s_t, a_t) \end{cases}$$

That is, given a state s_t, the action a_t only depends on the current state s_t, and the next state s_{t+1} only depends on the current state s_t and the current action a_t. The sequence is a follows:

$$s_t \to a_t \to (s_{t+1}, r(s_t, a_t))$$

The goal of the agent is to choose some actions in each state that optimises the value of some performance metric. These actions are stored in a policy. Hence, the agent is interested in the optimal policy. Given the sequential order in Statement (15.1.1), the policy is defined as follows:

Definition 15.3.2 *A policy π is the probability of an action a given the state s*

$$\pi(s, a) = P(a_t = a|s_t = s) = P(a|s)$$

Note, we will sometime denote $\pi(s_t, a_t)$ to represent the probability $P(a_t|s_t)$ when focusing on the variables (s_t, a_t). Further, to emphasise a parametric policy, we will denote $\pi_\theta(s, a) = P(a_t = a|s_t = s, \theta)$, where $\theta \in \mathbb{R}^l$, for $l << |\mathcal{S}|$, is a parameter vector.
A policy fully defines the behaviour of an agent. MDP policies depend on the current state and not the history. Thus, they are stationary:

$$a_t \sim \pi(\cdot|s_t)\,, \forall t > 0$$

The policy function π can be stochastic (see Definition (15.3.1)) or deterministic. That is, given the current state s_t, one chooses the action according to a stochastic or deterministic function, respectively.

- In the case where it is a deterministic function, it can behaves like a δ-function (there is a probability 1 to take an action $a^{'}$ given a state s, and a probability 0 to take all other actions). It represent the action chosen.
- On the other hand, if it is a stochastic function, it represent a probability (see Definition (15.3.1)). It is denoted by $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ where $\mathcal{P}(\mathcal{A})$ is the set of probability measures on $\mathcal{A}$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

More formally, we have:

Definition 15.3.3 *The MDP (S, A) is said to have a deterministic action if and only if its stochastic policy $\pi^{(sto)}$ is deterministic. That is, there exists a measurable function $\pi : \mathcal{S} \to \mathcal{A}$ such that*

$$\forall (B^{'}, s) \in \mathcal{B}(\mathcal{A}) \times \mathcal{S}\,,\ \pi^{(sto)}(B^{'} \mid s) = \begin{cases} 1 \text{ if } \pi(s) \in B^{'} \\ 0 \text{ otherwise} \end{cases}$$

π is the deterministic policy.

In that case, the action a_t is simply given by $a_t = \pi(s_t)$.

15.3.1.2 The transition probabilities

In the method of Markov chain (see Appendix (24.6.3)), we let $\mu_{t,i} = P(s_t = i)$ be a vector of probabilities, the mass function of s_t, and write μ_t for the raw vector with entries $(\mu_{t,i})$. We define the transition operator $\mathcal{T}$ as

$$\mathcal{T}_{i,j} = P(s_{t+1} = i | s_j = j)$$

so that $\mu_{t+1} = \mathcal{T}\mu_t$ (see Lemma (24.6.4)). We denote the transition probability as $P(s_{t+1}|s_t)$. Accounting for the policy, we let $\xi_{t,k} = P(a_t = k)$ and write the state-action transition operator as

$$\mathcal{T}_{i,j,k} = P(s_{t+1} = i | s_j = j, a_t = k)$$

so that

$$\mu_{t+1,i} = \sum_{j,k} \mathcal{T}_{i,j,k} \mu_{t,j} \xi_{t,k}$$

We get the relation

$$\begin{bmatrix} s_{t+1} \\ a_{t+1} \end{bmatrix} = \mathcal{T} \begin{bmatrix} s_t \\ a_t \end{bmatrix} \text{ and } \begin{bmatrix} s_{t+k} \\ a_{t+k} \end{bmatrix} = \mathcal{T}^k \begin{bmatrix} s_t \\ a_t \end{bmatrix}$$

For simplicity of exposition we denote the state transition probability as

$$P(s^{'}|s, a) = P(s_{t+1} = s^{'} | (s_t = s, a_t = a))$$

where the pair (s, a) has fixed values. That is, we omit to represent the variables and focus on the values. This is the probability that action a in state s at time t will lead to state $s^{'}$ at time $t + 1$.
The state and reward transition probability is

$$P(s^{'}, r^{'}|s, a) = P\big(s_{t+1} = s^{'}, r_{t+1} = r^{'} | (s_t = s, a_t = a)$$

This is the probability that action a in state s at time t will lead to state $s^{'}$ and reward $r^{'}$ at time $t + 1$. As a result, $\sum_{s' \in \mathcal{S}} \sum_{r' \in \mathcal{R}} P(s^{'}, r^{'}|s, a) = 1$. From the marginal probability mass function (see Lemma (24.1.4)), we can get the probability to arrive at a state $s^{'}$ by summing over the possible rewards, getting

$$\begin{aligned} P(s^{'}|s, a) &= \sum_{r' \in \mathcal{R}} P\big(s_{t+1} = s^{'}, r_{t+1} = r^{'} | (s_t = s, a_t = a)\big) \\ &= P(s_{t+1} = s^{'} | (s_t = s, a_t = a)) \end{aligned}$$

Following the sequential order in Statement (15.1.1) and, using the Markov property (see Appendix (24.6.3)), the joint probability distribution of T random variables $s_1, s_2, ..., s_T$ (see Equation (24.1.3)) simplifies to

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$P(s_1, s_2, ..., s_T) = P(s_1) \times P(s_2|s_1) \times \cdots \times P(s_T|s_{T-1})$$

Adding the action, we get

$$P(s_1, a_1, ..., s_T, a_T) = P(s_1) \prod_{t=1}^{T} \pi(s_t, a_t) P(s_{t+1}|(s_t, a_t)) \tag{15.3.1}$$

We also have the relation

$$P(s_{t+1} = s^{'}, a_{t+1} = a^{'}|(s_t = s, a_t = a)) = P(s_{t+1} = s^{'}|(s_t = s, a_t = a)) P(a_{t+1} = a^{'}|s_{t+1} = s^{'})$$

Computing expectation by conditioning (see Equation (24.1.9)), we get the immediate average reward, given a state and an action, as follows:

$$\begin{aligned} R(s, a) &= E[r_{t+1}|(s_t = s, a_t = a)] \\ &= \sum_{r^{'} \in \mathcal{R}} r^{'} P(s^{'}, a^{'}|(s, a)) \end{aligned} \tag{15.3.2}$$

It is the immediate reward received after transitioning from state s to state $s^{'}$, due to action a.
The state and reward sequence $(s_1, r_2), (s_2, r_3), ...$ is a Markov reward process $(\mathcal{S}, P_\pi, \mathcal{R}_\pi, \gamma)$ where the state transition probability, under the policy π, is

$$\begin{aligned} P_\pi(s^{'}|s) &= \sum_{a \in \mathcal{A}} P(a_t = a|s_t = s) P(s_{t+1} = s^{'}|(s_t = s, a_t = a)) \\ &= \sum_{a \in \mathcal{A}} P(a|s) P(s^{'}|(s, a)) \end{aligned}$$

Also, using the conditional expectation, under the policy π, we get

$$\begin{aligned} R_{\pi,s} = E_\pi[r_{t+1}|s_t = s] &= \sum_{a \in \mathcal{A}} P(a_t = a|s_t = s) E[r_{t+1}|(s_t = s, a_t = a)] \\ &= \sum_{a \in \mathcal{A}} P(a|s) \sum_{r^{'} \in \mathcal{R}} r^{'} P(r^{'}|s, a) \\ &= \sum_{a \in \mathcal{A}} P(a|s) R(s, a) \end{aligned}$$

Similarly, for a smooth function $f(\cdot)$, we get

$$\begin{aligned} F_{\pi,s} = E_\pi[f(s_{t+1})|s_t = s] &= \sum_{a \in \mathcal{A}} P(a_t = a|s_t = s) E[f(s_{t+1})|(s_t = s, a_t = a)] \\ &= \sum_{a \in \mathcal{A}} P(a_t = a|s_t = s) \sum_{s^{'} \in \mathcal{S}} f(s^{'}) P(s_{t+1} = s^{'}|(s_t = s, a_t = a)) \\ &= \sum_{a \in \mathcal{A}} P(a|s) \sum_{s^{'} \in \mathcal{S}} f(s^{'}) P(s^{'}|s, a) \\ &= \sum_{s^{'} \in \mathcal{S}} f(s^{'}) \sum_{a \in \mathcal{A}} P(a|s) P(s^{'}|s, a) \\ &= \sum_{s^{'} \in \mathcal{S}} f(s^{'}) P_\pi(s^{'}|s) \end{aligned} \tag{15.3.3}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We can summarise the MDP as follows:

- Environment (E): $P(s^{'}|s,a)$ is the state transition matrix and $R(s,a) =< r|s,a >$ is the conditional average reward.
- Agent (A): $\pi(s,a) = P(a|s)$ is the policy.

15.3.1.3 Performance metrics

In order to maximise future rewards, we need to find a way of defining the average reward as we do not want the sum of all future rewards to be infinite (this is the assumption in Equation (15.4.26)). To do so we define the discount factor as follows: if one earn x dollars at time τ, the current value of the x dollars will be

$$x(\frac{1}{1+r})^{\tau}$$

where r is the interest rate. We denote $\gamma = \frac{1}{1+r}$. When the transition in the MDP is over one time period, then $\tau = 1$ and the discount factor is just γ. We need the following assumption:

Assumption 3 *For every policy π, the sum of the discounted rewards is integrable. That is,*

$$\forall s \in \mathcal{S}\ ,\ E_{\pi}[\sum_{t=0}^{\infty}|\gamma^t r(s_t,\pi(s_t),s_{t+1})||s_0 = s] < \infty$$

Several solutions exists such as:

- Fixed finite horizon (total reward): The expected total reward over a finite horizon of T time periods for a policy π, when starting at state i, is

  $$R_{\pi}(s) = E_{\pi}[\sum_{t=1}^{T} r(s_t,\pi(s_t),s_{t+1})|s_1 = s]$$

  where s_t is the state at time t and the starting state s is fixed for the problem.
  At time t, we want to maximise the total reward

  $$R_t = r_{t+1} + r_{t+2} + \cdots + r_T = \sum_{k=0}^{(T-t)-1} r_{t+1+k}$$

- Terminal state: The overall time may vary but it remains finite with probability 1. It is achieved in MDP with a terminal state denoted g, so that the following conditions apply

  $$\begin{aligned} P(s^{'}|g,a) &= \delta_{s^{'},g} \\ P(s^{'},r^{'}|g,a) &= \delta_{r,0} \\ \lim_{t\to\infty} P(s_t = g) &= 1\ ,\forall\pi \end{aligned}$$

  In this setting, the goal is to maximise the expectation of

  $$R_t = \sum_{k=0}^{\infty} r_{t+1+k}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Discounted reward: The expected total discounted reward for a policy π, starting at state s, over an infinitely long time horizon is

$$R_\pi(s) = \lim_{K\to\infty} \inf E_\pi[\sum_{k=1}^{K} \gamma^{k-1} r(s_k, \pi(s_k), s_{k+1})|s_1 = s]$$

At time t, the discounted reward (DR) is

$$\begin{aligned} R_t &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \\ &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \end{aligned} \tag{15.3.4}$$

where $0 < \gamma < 1$ is the discount rate. It is using the properties of geometric series [2] For small γ we ignore farther-away events, and vice versa for large γ.
Returns at successive time steps are related to each other in an iterative way. We write the rate R_t as

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \\ &= r_{t+1} + \gamma\big(r_{t+2} + \gamma r_{t+3} + \cdots\big) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned} \tag{15.3.5}$$

We can multiply the DR with α, getting

$$\alpha R_t = \frac{1}{\text{norm}}\big(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots\big)$$

where $\alpha = \frac{1}{\text{norm}}$ and norm $= (1-\gamma)$.

- Average reward: The expected average reward for a policy π per transition, starting at state s, over an infinitely long time horizon is

$$R_\pi(s) = \lim_{K\to\infty} \inf \frac{1}{K} E_\pi[\sum_{k=1}^{K} r(s_k, \pi(s_k), s_{k+1})|s_1 = s]$$

In the case of a regular policy, the average reward does not depend on the starting state, that is, $R_\pi(s) = R_\pi$ for all s. The goal becomes to maximise the average reward.

15.3.1.4 The value function

Note, the total reward R_t is a random variable, where the reward r_t at each stage is stochastic, depending on the behaviour of the environment and on the policy of the agent. Hence, we want to maximise the mean reward R_t. The main principle of dynamic programming (DP) is to solve the credit assignment problem by constructing an evaluation function called the value function on the state-space. The value function $V(s)$ of a state s assigns a (long-term) value to each state. It is the average discounted reward, under the policy π, given that we start at state s. It is given by

$$V_\pi(s) = E_\pi[R_t|s_t = s] =< R_t|s_t = s > \ , \forall s \in \mathcal{S}$$

[2] The geometric series states that for $|x| < 1$ we have $\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n$ which is an infinite series with finite sums. The telescoping series states that for $|x| < 1$ we define $g(K) = \frac{x^K}{1-x}$, then $g(K) \to 0$ as $K \to \infty$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where R_t is defined in Section (15.3.1.3). $E_\pi[\cdot]$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step. Thus, the value function depends on the policy of the agent. Note, it corresponds to the performance function in Equation (15.4.25), in the continuous case. The dynamics of MDP will cause the agent to wander in the state space, so that we need to be able to maximise the value function from all initial conditions.

15.3.2 The Bellman equations

A fundamental property of value functions used in dynamic programming (and reinforcement learning) is that they satisfy recursive relationships. The Bellman equations (see Bellman [1954], [1957]) are a set of linear equations having a unique solution, for $0 < \gamma < 1$ (under mild conditions on the transition matrix). Thus, it can be solved by matrix inversion. They were presented in the form of the value functions.

15.3.2.1 Description

We assume that we use the discounted reward defined in Equation (15.3.5) along with the Markov property. Using conditional expectation in Equation (15.3.2), for any policy π and any state s, we can rewrite the value function into two parts as

$$\begin{aligned} V_\pi(s) &= < R_t | s_t = s > \\ &= < r_{t+1} + \gamma R_{t+1} | s_t = s > \\ &= < r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s > \\ &= \sum_{a \in \mathcal{A}} \pi(s,a) \Big(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_\pi(s') \Big) \text{ for each } s \in \mathcal{S} \end{aligned} \tag{15.3.6}$$

where $\pi(s,a)$ is the policy. From the definition of R and P in Section (15.3.1.2), this expression is a sum over all values of the three variables a, s', and r'. For each triple, we compute the probability $\pi(s,a)P(s',r'|s,a)$, which is used to weight the quantity $\big(r + \gamma V_\pi(s')\big)$. We then sum over all possibilities to get an expected value. We are looking ahead from a state s to its possible successor states. Each open circle represents a state and each solid circle represents a state-action pair. This is illustrated with the backup diagram in Figure (15.2). Starting from state s at the top of the tree, the agent can take any of some set of actions based on policy π. It leads to several next state s' along with a reward r' depending on the dynamics given by P. The Bellman equation averages over all the possibilities, weighting each of them with its probability of occurrence. It states that the value of the start state must be equal to the discounted value of the expected next state together with the reward expected along the path.

Figure 15.2: Backup diagram for the value function.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Given both the conditional expectation and the state transition probability, under the policy π, in Section (15.3.1.2), we can rewrite the value function as

$$V_\pi(s) = R_{\pi,s} + \gamma \sum_{s^{'} \in \mathcal{S}} P_\pi(s^{'}|s) V_\pi(s^{'}) \text{ for each } s \in \mathcal{S}$$

Since R and P are known, the evaluation function is computed by solving a set of linear equations, one for each state. In the simple case where the action a is not a random variable (MRP), the above equation simplifies to

$$V(s) = R(s) + \gamma \sum_{s^{'} \in \mathcal{S}} P(s^{'}|s) V(s^{'})$$

where $R(s) =< r_{t+1}|s_t = s >$. Note, there is no reference to the policy π.
In the case where the action a is not a random variable, the Bellman equation can be expressed using matrices as

$$V_s = R_s + \gamma T_{ss^{'}} V_s \tag{15.3.7}$$

where $V_s = \big(V(1), ..., V(n)\big)^\top$ is a column vector with one entry per state, $R_s = R(s)$ is a column vector of size n, and $T_{ss^{'}} = P(s^{'}|s)$ is an $n \times n$ transition matrix. That is,

$$\begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} T_{11} & \cdots & T_{1n} \\ \vdots & & \vdots \\ T_{n1} & \cdots & T_{nn} \end{bmatrix} \begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix}$$

The value function becomes

$$V_s = \big(I - \gamma T_{ss^{'}}\big)^{-1} R_s$$

In the case where the action is a random variable (the MDP), the Bellman equation can be expressed as

$$V_\pi = R_{\pi,s} + \gamma T_{\pi,ss^{'}} V_\pi \tag{15.3.8}$$

where

$$T_{\pi,ss^{'}} = \sum_{a \in \mathcal{A}} \pi(s,a) P(s^{'}|s,a) = P_\pi(s^{'}|s) \text{ and } R_{\pi,s} = \sum_{a \in \mathcal{A}} \pi(s,a) R(s,a)$$

and the value function becomes

$$V_\pi = \big(I - \gamma T_{\pi,ss^{'}}\big)^{-1} R_{\pi,s} \tag{15.3.9}$$

15.3.2.2 Approximation

Since the computational cost of inverting matrices in Equation (15.3.9) is large: a number of order $O(n^3)$, where n is the number of states, one solution is to use an iterative approximation for the value function V_π. Some iterative methods for large Markov reward process (MRP) and Markov decision process (MDP) are:

- Dynamic programming
- Monte Carlo evaluation
- Temporal difference (TD) learning

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

When the return $R(s,a)$ and the transition probability $P(s^{'}|s,a)$ are known, the Iterative Policy Evaluation is an example of Dynamic Programming Algorithm which uses the Bellman equation (see Equation (15.3.6)) as an update rule for successive approximations $V_0, V_1, V_2, ...$ of the true value function. We can start from arbitrary initial function $V_0(s)$ and compute the $k+1$ approximation as

$$V_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \Big(R(s,a) + \gamma \sum_{s^{'} \in \mathcal{S}} P(s^{'}|s,a) V_k(s^{'}) \Big)$$

Theorem 15.3.1 *The series of approximations converge asymptotically to $V_\pi(s)$.*

We will describe these methods in details in Section (17.1).

15.3.3 A mathematical formalism

15.3.3.1 Overview

15.3.3.1.1 Formulation of the problem We consider the standard reinforcement learning framework of a Markov decision problem (MDP) (see Section (15.3.1)). A (stochastic) policy π (see Definition (15.3.2)) is a function from $\mathcal{B}(\mathcal{A}) \times \mathcal{S}$ to $[0,1]$, $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, that specifies what action to take in each state. The cumulative discounted reward is given by

$$R_t = \sum_{t \geqslant 0} \gamma^t r_t \, , r_t = r(s_t, a_t) \tag{15.3.10}$$

Note, this formulation of the cumulative discounted reward is slightly different from $R_t = \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k}$ where the first element is r_{t+1}.
As discussed by Sutton et al. [2000], there are two useful ways of formulating the objective of an agent: (1) the start-state formulation, (2) the average-reward formulation.
In the start-state formulation, we start from the state s_0 and want to get the long-term reward from that point. We need to deal with the randomness from the initial state $s_0 \sim p(s_0)$, the action $a_t \sim \pi(\cdot|s_t)$, the next state $s_{t+1} \sim p(\cdot|s_t, a_t)$ and the transition probability. To do so, we let the objective $J(\pi)$ be the expected cumulative discounted reward

$$J(\pi) = E[\sum_{t \geqslant 0} \gamma^t r_t | \pi]$$

which we maximise to find the optimum policy as follows:

$$\pi^* = \arg\max_{\pi} J(\pi)$$

Thus, we get the value function and action-value function as

$$V_\pi = E[\sum_{t \geqslant 0} \gamma^t r_t | \, s_0, \pi] \text{ and } Q_\pi(s,a) = E[\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} | \, s_t = s, a_t = a, \pi]$$

The backup diagram for the action-value function is given in Figure (15.3).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 15.3: Backup diagram for the action-value function.

We let $d_\pi(s)$ be the discounted weighting of states obtained by starting at s_0 and then following π. It is given by

$$d_\pi(s) = \sum_{t=0}^{\infty} \gamma^t P_\pi(s_t = s|\ s_0)$$

Changing slightly notation, we define the (improper) discounted state distribution as

$$\rho_\pi(s^{'}) = \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t p_0(s) p(s^{'}|s,a) ds$$

We can then rewrite the objective as

$$\begin{aligned} J_\pi &= \int_{\mathcal{S}} \rho_\pi(s) \int_{\mathcal{A}} \pi(s,a) R(s,a) da ds \\ &= E_{s\sim\rho_\pi, a\sim\pi}[R(s,a)] \end{aligned}$$

From Algorithm (30) we see that following a policy, π, produces the trajectories (or paths)

$$\tau = (s_0, a_0, r_0), (s_1, a_1, r_1), ...$$

over $\mathcal{S} \times \mathcal{A} \times \mathbb{R}$. By analogy with stochastic control, these trajectories correspond to the state of the system, and the action a corresponds to the stochastic control u.
Assuming a fixed maturity T and setting $\gamma = 1$, from Equation (15.3.1), we can write the joint mass function as

$$\begin{aligned} p(s_0, a_0, s_1, a_1, ..., s_T, a_T) = p(\tau) &= \prod_{t\geqslant 0} P(s_{t+1}|(s_t, a_t))\pi(s_t, a_t) \\ &= p(s_0)\pi(s_0, a_0) \prod_{t\geqslant 1} P(s_{t+1}|(s_t, a_t))\pi(s_t, a_t) \end{aligned} \tag{15.3.11}$$

where τ is a trajectory. We can rewrite the optimisation problem with respect to the density $p(\tau)$ as

$$\pi^* = \arg\max_\pi E_{\tau\sim p(\tau)}[\sum_{t\geqslant 0} r(s_t, a_t)]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $E_{\tau\sim p(\tau)}[\cdot]$ indicates that actions are sampled from π to generate the trajectory $\tau = (s_0, a_0, s_1, a_1, ...)$. The expectation operator being a linear operator, the optimisation problem can be written as

$$\pi^* = \arg\max_{\pi} \sum_{t\geqslant 0} E_{(s_t,a_t)\sim p(s_t,a_t)}[r(s_t, a_t)]$$

This expectation is expressed with respect to the state-action pair (s_t, a_t) and the joint probability distribution $p(s_t, a_t)$. In the case of finite horizon, we have

$$\pi^* = \arg\max_{\pi} \sum_{t=1}^{T} E_{(s_t,a_t)\sim p(s_t,a_t)}[r(s_t, a_t)]$$

If $T = \infty$ we want the state-action marginal $p(s_t, a_t)$ to converge to a stationary distribution. That is, for $\mu = p(s, a)$, we want $\mu = \mathcal{T}\mu$. Thus, in the limit $T \to \infty$, we get

$$\pi^* = \arg\max_{\pi} \frac{1}{T} \sum_{t=1}^{T} E_{(s_t,a_t)\sim p(s_t,a_t)}[r(s_t, a_t)] \to E_{(s,a)\sim p(s,a)}[R(s, a)]$$

In the average-reward formulation, we get the long-term expected reward per step as

$$\rho(\pi) = \lim_{T\to\infty} \frac{1}{T} E[\sum_{t=1}^{T} r(s_t, a_t)|\, \pi] = \sum_{s\in\mathcal{S}} d_\pi(s) \sum_{a\in\mathcal{A}} \pi(s, a)R(s, a)$$

where $d_\pi(s) = \lim_{t\to\infty} P_\pi(s_t = s|\, s_0)$ is the stationary distribution of states under policy π. We want to find a policy π^* that maximise the average reward:

$$\pi^* = \arg\max_{\pi} \rho(\pi)$$

In that setting, a policy π is assessed based on the expected differential reward associated with states s or state-action pair (s, a). The differential value of a state-action pair, given a policy π, is

$$Q_\pi(s, a) = \sum_{t=1}^{\infty} E[r_t - \rho(\pi)|\, s_0 = s, a_0 = a, \pi]\,, \forall s \in \mathcal{S}, a \in \mathcal{A}$$

and the differential value function is

$$V_\pi(s) = \sum_{a\in\mathcal{A}} \pi(s, a)Q_\pi(s, a)$$

15.3.3.1.2 Solving the problem Following the sequential order in Statement (15.1.1), we have $s_t \sim p(s_t)$ and $a_t \sim \pi(s_t, a_t)$, so that the expectation can be decomposed as

$$E_{\tau\sim p(\tau)}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)] = \sum_{t=0}^{\infty} E_{s_t\sim p(s_t)}[E_{a_t\sim\pi(a_t,s_t)}[\gamma^t r(s_t, a_t)]]$$

which is a sum of conditional expectations.
If we write the future reward from time $t = 1$ as

$$Q(s_1, a_1) = r(s_1, a_1) + E_{s_2\sim p(s_2|s_1,a_1)}[E_{a_2\sim\pi(a_2,s_2)}[r(s_2, a_2) + \cdots |s_2]\,|(s_1, a_1)]$$

where $s_{t+1} \sim p(s_{t+1}|\, s_t, a_t)$, then we get

$$E_{s_1\sim p(s_1)}[E_{a_1\sim\pi(a_1,s_1)}[Q(s_1, a_1)]\,|s_1]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In the special case where the policy is deterministic (see Definition (15.3.3)), it can be described by the function $\nu : \mathcal{S} \to \mathcal{A}$. Setting $a_1 = \nu(s_1)$, the above equation simplifies to

$$E_{s_1 \sim p(s_1)}[Q(s_1, \nu(s_1)) \,|s_1]$$

If we know $Q(s_1, a_1)$, we can modify the policy $\pi(a_1, s_1)$ to improve it further.

- For example, a well known policy (the greedy policy) is

$$\pi(a_1, s_1) = 1 \text{ if } a_1 = \arg\max_{a_1} Q(s_1, a_1)$$

  and zero otherwise.

- Alternatively, we can compute the gradient to increase the probability of getting good actions.

If we define the average $V_\pi(s) = E_\pi[Q_\pi(s, a)]$, and if $Q_\pi(s, a) > V_\pi(s)$, then the action is better than average. Consequently, we can modify π to improve the action. It leads to the approach of the value function and Q-value function which we are now going to present.

15.3.3.2 The value function

Recall, $\mathcal{S}$ and $\mathcal{A}$ are countable.

Definition 15.3.4 *The value function of an MDP at state s is the expected cumulative reward from following the policy from state s:*

$$V_\pi(s) = E[\sum_{t \geqslant 0} \gamma^t r_t | s_0 = s, \pi]$$

Note, this function corresponds to the performance function in Equation (15.4.25) with $K(\cdot) = 0$, the continuous integral is replaced by a discrete one, and the utility function is linear with respect to the reward r.

***Remark* 15.3.1** *The value function is conditioned to the state s (fixed). The action is a random variable.*

Thus, using the conditional expectation (see Equation (15.3.3)), we can rewrite the value function as

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(s, a) E_\pi[\sum_{t \geqslant 0} \gamma^t r_t | (s_0 = s, a_0 = a)] \tag{15.3.12}$$

As discussed in Section (15.3.2), the value function can be decomposed into immediate reward plus discounted value of successor states

$$V_\pi(s) = E_\pi[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s] \tag{15.3.13}$$

which we can write as

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(s, a) \Big(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V_\pi(s') \Big) \tag{15.3.14}$$

These two relations are called the Bellman equation (see Section (15.3.2)).

***Proposition* 4** *Suppose that $\gamma < 1$ and V_π is bounded. Then V_π is the unique solution to the Bellman equation in $L^\infty(\mathcal{S})$ the space of bounded and measurable functions from $\mathcal{S}$ to $\mathbb{R}$.*

The learning goal is to obtain the best possible value for each initial state, that is, to obtain the optimal value function.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 15.3.5 *The optimal value function V^* is the maximum expected cumulative reward achievable from a given state over all policies:*

$$V^*(s) = \max_{\pi} V_{\pi}(s)$$

An MDP is solved once we know the optimal value function V^*. Thus, we need to find π^* such that

$$V_{\pi*}(s) = V^*(s)\,, \forall s$$

Such a policy is an optimal policy since it satisfies

$$V_{\pi*}(s) \geqslant V_{\pi}(s)\,, \forall \pi, s$$

As discussed in Appendix (15.4), under some conditions, such a policy exists.
We can calculate the value of $\pi(s)$ within the value function $V(s)$ whenever it is needed. In that case, the optimal value function $V^*(s)$ satisfies the Bellman optimality equation (BOE) (value function in Appendix (28.3.23))

$$V^*(s) = \max_{a \in \mathcal{A}} \{R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)V^*(s')\}\,, \forall s \tag{15.3.15}$$

If $\mathcal{S}$ and $\mathcal{A}$ are finite set, and $0 < \gamma < 1$, then the above equation has a unique solution. The backup diagram for the BOE is given in Figure (15.4).

Figure 15.4: Backup diagram for the BOE.

15.3.3.2.1 The optimal policy The optimal policy, satisfying $V_{\pi*} = V^*$, is the deterministic policy (see Definition (15.3.3)) given by

$$\pi(s,a) = \delta_{a,a(s)} \tag{15.3.16}$$

where $\delta_{a,b} = 1$ if $a = b$, and zero otherwise, and

$$a(s) = \arg\max_{a \in \mathcal{A}} \{R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)V^*(s')\}\,, \forall s \tag{15.3.17}$$

Proof: To prove the existence of a solution to the BOE we start with a policy $\pi \Rightarrow V_{\pi}$. We then define a new policy π' as follows:

- π' is deterministic: $\pi'(s,a) = \delta_{a,a(s)}$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- $a(s) = \arg\max_a \{R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)V_\pi(s')\}$

We use that new policy to obtain a new value function $V_{\pi'}(s)$. Then, we need to prove that $V_{\pi'}(s) \geqslant V_\pi(s)$, for all s. We have

$$V_\pi(s) \leqslant E_{\pi'}[r_{t+1} + \gamma V_\pi(s_{t+1})|s_t = s], \forall s$$

which implies

$$\begin{aligned}
V_\pi(s) &\leqslant E_{\pi'}[r_{t+1} + \gamma V_\pi(s_{t+1})|s_t = s] \\
&= E_{\pi'}[r_{t+1} + \gamma E_\pi[r_{t+2} + \gamma V_\pi(s_{t+2})]|s_t = s] \\
&\leqslant E_{\pi'}[r_{t+1} + \gamma E_{\pi'}[r_{t+2} + \gamma V_\pi(s_{t+2})]|s_t = s] \\
&= E_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 V_\pi(s_{t+2})|s_t = s] \\
&\leqslant E_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots|s_t = s] \\
&= V_{\pi'}(s)
\end{aligned}$$

so that the new policy improves the previous one. At each step, we move in the most greedy manner to improve the policy. This iterative process is called Greedy Policy Improvement. Allowing for a finite number of states (s,a), then $V_\pi(s)$ is bounded from above, allowing for convergence to a finite value. Thus, $\pi \to \pi^\infty$ then $V_\pi \to V_\infty$ which satisfy the Bellman optimality equation. Further, $V_\infty = V_{\pi^\infty}$, where

$$\pi^\infty(s,a) = \delta_{a,a(s)}$$

and

$$a(s) = \arg\max_a \{R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)V_\infty(s')\}$$

Note, $V_\infty(s)$ can be a local maximum which also satisfies the BOE. One can prove that there is a single value function satisfying that equation.

15.3.3.3 The Q-value function

Similarly to the value function, which assigns values to states, we can also write a value function of states and actions. It is called the Q-value function.

Definition 15.3.6 *The Q-value function at state s and action a is the expected cumulative reward from taking action a in state s and then following the policy π:*

$$Q_\pi(s,a) = E[\sum_{t \geqslant 0} \gamma^t r_t|(s_0 = s, a_0 = a), \pi]$$

***Remark* 15.3.2** *The Q-value function summarises the performance of each action from a given state, assuming it follows π afterwards. It is conditioned to the fixed pair (s,a). There is no need to sum over all possible actions.*

***Remark* 15.3.3** *Williams [1992] suggested that $Q_\pi(s,a)$ could be approximated with the actual returns $R_t = \sum_{t \geqslant 0} \gamma^t r_t$ (see Section (17.1.4.1)). This is a property of ergodic processes (see Appendix (23.4.2)).*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The Q-value function can be decomposed into immediate reward plus discounted value of successor states

$$Q_\pi(s,a) = E_\pi[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1})|(s_t = s, a_t = a)] \tag{15.3.18}$$

which we can write as

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s^{'} \in \mathcal{S}, a^{'} \in \mathcal{A}} P(s^{'}|(s,a))\pi(s^{'}, a^{'})Q_\pi(s^{'}, a^{'})$$

or more simply as

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s^{'} \in \mathcal{S}, a^{'} \in \mathcal{A}} P(s^{'}, a^{'}|(s,a))Q_\pi(s^{'}, a^{'})$$

Further, given Equation (15.3.12), we can write the value function $V(s)$ in terms of the Q-value function as

$$\begin{aligned} V_\pi(s) &= \sum_{a \in \mathcal{A}} \pi(s,a)Q_\pi(s,a) \\ &= E_{a_t \sim \pi(s_t, a_t)}[Q_\pi(s_t, a_t)] \end{aligned} \tag{15.3.19}$$

so that the Q-value function becomes

$$\begin{aligned} Q_\pi(s,a) &= R(s,a) + \gamma \sum_{s^{'} \in \mathcal{S}} P(s^{'}|s,a)V_\pi(s^{'}) \\ &= R(s,a) + \gamma E_{s_{t+1} \sim p(s_{t+1}|(s_t, a_t))}[V_\pi(s_{t+1})] \end{aligned} \tag{15.3.20}$$

Combining Equation (15.3.20) with Equation (15.3.19), we can also write the Q-value function as

$$Q_\pi(s_t, a_t) = R(s_t, a_t) + \gamma E_{s_{t+1} \sim p(s_{t+1}|(s_t, a_t))}[E_{a_{t+1} \sim \pi(s_{t+1}, a_{t+1})}[Q_\pi(s_{t+1}, a_{t+1})]] \tag{15.3.21}$$

In the special case where the policy is deterministic (see Definition (15.3.3)), it can be described by the function $\nu : \mathcal{S} \to \mathcal{A}$. Setting $a_{t+1} = \nu(s_{t+1})$, the above equation simplifies to

$$Q_\nu(s_t, a_t) = R(s_t, a_t) + \gamma E_{s_{t+1} \sim p(s_{t+1}|(s_t, a_t))}[Q_\nu(s_{t+1}, \nu(s_{t+1}))] \tag{15.3.22}$$

***Remark* 15.3.4** *Unless explicitly stated, we will use the notation π for the deterministic policy.*

The relation between the value function and the Q-value function leads to the Policy Improvement Theorem. It is given as follows:

Theorem 15.3.2 *Suppose that either $\gamma < 1$ and V_π is bounded on $\mathcal{S}$ or the MDP has finite horizon. Given the policy π, let $\pi^{'}$ be another policy. We have*

$$\big(\forall s \in \mathcal{S}\,,\, E_{\pi'}[Q_\pi(s,a_t)|\, s_t = s] \geqslant V_\pi(s)\big) \Longrightarrow \big(\forall s \in \mathcal{S}\,,\, V_{\pi'}(s) \geqslant V_\pi(s)\big)$$

Proof: see Ross [1996].
If the inequality is strict for at least one state on the LHS, then the inequality on the RHS is also strict for at least one state. The policy $\pi^{'}$ is said to be better than π if and only if the inequality on the RHS is satisfied. A particular policy is always better than the previous one. This policy improvement is easy to set if Q_π is known and $\mathcal{A}$ is small.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 15.3.7 *Assuming that $\mathcal{A}$ is finite, the deterministic policy $\pi^{'}$ defined by*

$$\pi^{'}(s) = \arg\max_{a \in \mathcal{A}} Q_{\pi}(s, a)$$

is called a greedy policy.

Given Theorem (15.3.2), this policy is better than π. We therefore have a method for optimising a policy. The boundedness condition is satisfied if $\mathcal{S} \times \mathcal{A}$ is finite.
The following proposition gives a condition on policy optimality:

Proposition 5 *Suppose that either $\gamma < 1$ and V_{π} is bounded on $\mathcal{S}$ or the MDP has finite horizon. Then, π is optimal if and only if*

$$\forall (s, a) \in \mathcal{S} \times \mathcal{A}\,,\, \pi(s, a) \geqslant 0 \Longrightarrow a \in \arg\max_{a^{'} \in \mathcal{A}} Q_{\pi}(s, a^{'})$$

The proposition states that if the policy can no-longer be improved (using the greedy policy method), then the policy is an optimal policy. In the case where $\mathcal{S} \times \mathcal{A}$ is finite, we get the following result:

Proposition 6 *Suppose that $\gamma < 1$ and $\mathcal{S} \times \mathcal{A}$ is finite. Then, there exists a deterministic optimal policy π^*.*

***Remark* 15.3.5** *The optimal policy π^* corresponds to taking the best action in any state as specified by Q^*. Thus, we want a greedy policy π^* that always seeks an action maximising the Q-value function in the current state:*

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a)$$

We now states some results about the best Q-value function Q^*.

Definition 15.3.8 *The optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a given $(state, action)$ pair over all policies:*

$$Q^*(s, a) = \max_{\pi} Q_{\pi}(s, a)$$

Thus, we need to find π^* such that

$$Q_{\pi*}(s, a) = Q^*(s, a)$$

Since it is easier to work with Q than V, then given Q^*, we can infer V^* (see Equation (15.3.15)) as follows:

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a) \tag{15.3.23}$$

which is

$$V^*(s) = \max_{a \in \mathcal{A}} \big(R(s, a) + \gamma \sum_{s^{'} \in \mathcal{S}} P(s^{'}|s, a) V^*(s^{'})\big)$$

Putting terms together, we can write the optimal Q-value function as

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s^{'} \in \mathcal{S}} P(s^{'}|s, a) \max_{a^{'} \in \mathcal{A}} Q^*(s^{'}, a^{'})$$

The policy-improvement algorithm guarantees finding the optimal policy in a finite Markov decision problem. We can then define the Bellman optimal equation as follows:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Proposition 7 *Let $\gamma \in [0,1[$ and suppose that $\mathcal{A}$ is finite. Then the function $\psi : L^{\infty}(\mathcal{S} \times \mathcal{A}) \rightarrow L^{\infty}(\mathcal{S} \times \mathcal{A})$ defined by the expression*

$$\forall (Q,s,a) \in L^{\infty}(\mathcal{S} \times \mathcal{A}) \times \mathcal{S} \times \mathcal{A}\,,\, \psi(Q)(s,a) = r(s,a) + \gamma \int_{s' \in \mathcal{S}} \max_{a' \in \mathcal{A}} Q(s^{'},a^{'}) P(ds^{'}|s,a)$$

has unique fixed point Q_{∞}. Q_{∞} is said to satisfy the Bellman optimal equation (BOE).

Proof: This result is a consequence of the fixed point theorem (see Theorem (23.2.4)) since ψ is γ=contracting and $L^{\infty}(\mathcal{S} \times \mathcal{A})$ is complete (see Theorem (23.2.5)). Let $(Q_1, Q_2, s, a) \in L^{\infty}(\mathcal{S} \times \mathcal{A})^2 \times \mathcal{S} \times \mathcal{A}$. Let $a^*(s)$ denote the action maximising Q_1 knowing s. Then, we have

$$\begin{aligned} \psi(Q_1)(s,a) - \psi(Q_2)(s,a) &\leqslant \gamma \int_{s' \in \mathcal{S}} \big(Q_1(s^{'},a^*(s^{'})) - Q_2(s^{'},a^*(s^{'}))\big) P(ds^{'}|s,a) \\ &\leqslant \gamma \int_{s' \in \mathcal{S}} \|Q_1 - Q_2\|_{\infty} P(ds^{'}|s,a) = \gamma \|Q_1 - Q_2\|_{\infty} \end{aligned}$$

This inequality shows that ψ is γ-contracting, and then proves the result.

Proposition 8 *Suppose that $\gamma < 1$ and $\mathcal{A}$ is finite. Suppose that there exists an optimal policy π^* such that $Q_{\pi^*} = Q^*$. From the above equation and the principle of dynamic programming (see Appendix (28.3.3.1)), Q^* is the unique element in $L^{\infty}(\mathcal{S} \times \mathcal{A})$ satisfying the Bellman optimal equation (BOE) in Proposition (7). It can be written as*

$$Q^*(s,a) = E_{s' \sim p(s'|s,\pi(s))}[r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q^*(s^{'},a^{'})|s,a] \tag{15.3.24}$$

Proof: From Equation (15.3.18), we have

$$\forall (s,a) \in \mathcal{S} \times \mathcal{A}\,,\, Q^*(s,a) = E_{\pi^*}[r(s,a) + \gamma Q^*(s^{'},a^{\diamond})|s,a]$$

However, according to Proposition (5), and since π^* is optimal, we have

$$a^{\diamond} \in \arg\max_{a' \in \mathcal{A}} Q^*(s^{'},a^{'})$$

almost surely. From a mathematical argument discussed in Section (17.1.3.6), we have

$$\max_a Q(s,a) = Q\big(s, \arg\max_a Q(s,a)\big)$$

It yields the expected result. EOP.
It states that if the optimal state-action values for the next time step $Q^*(s^{'},a^{'})$ are known, then the optimal strategy is to take the action that maximises the expected value of $r + \gamma Q^*(s^{'},a^{'})$. The backup diagram for the BOE is given in Figure (15.5).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 15.5: Backup diagram for the BOE.

15.3.4 Approximation

Reinforcement learning is composed of two problems:

1. Policy evaluation (or prediction problem): It is the problem of estimating the value function V_π for a given policy π.
2. Control problem: It is the problem of finding the optimal policy. DP, TD, and Monte Carlo methods all use some variation of generalised policy iteration (GPI). The differences in the methods are mainly differences in their approaches to the prediction problem.

Several reinforcement learning algorithms exist for solving the optimal policy problem. Some dynamic programming solutions require the knowledge of the state transition function P and the reward function R. Others can solve for the optimal policy of an MDP using experimentation alone.

15.3.4.1 Known transition function and reward

We consider the case where P and R are given, and we seek the optimal policy π^* that maximises the expected discounted reward. The standard family of algorithms calculating this optimal policy requires storing two arrays indexed by state:

- The value V, which contains real values.
- The policy π, which contains actions.

At the end of the algorithm, π will contain the solution and $V(s)$ will contain the discounted sum of the rewards to be earned (on average) by following that solution from state s. The algorithm has two steps:

1. a value update, and
2. a policy update,

which are repeated in some order for all the states until no further changes take place. This is illustrated in Figure (15.6).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 15.6: Iterative Policy

Both steps recursively update a new estimation of the state value (see Equation (15.3.14)) and the optimal policy (see Equation (15.3.16)), by using an older estimation of those values. Their order depends on the variant of the algorithm. One can also estimate their values for all states at once, or state by state, and more often to some states than others. As long as no state is permanently excluded from either of the steps, the algorithm will eventually arrive at the correct solution. The generalised policy iteration (GPI) let the policy-evaluation and policy-improvement processes interact in an independent fashion. The evaluation and improvement processes can be understood as both competing (they pull in opposite directions) and cooperating. When both the evaluation process and the improvement process have converged, then the value (action-value) function and the policy must be optimal. Both processes only converge when a policy has been found that is greedy with respect to its own evaluation function.
Assuming full knowledge of the MDP, the two basic approaches for computing the optimal value function and optimal action-value function are Value Iteration (VI) (see Bellman [1957]) and Policy Iteration (PI) (see Howard [1960]).

- In VF, the principle is to solve the optimality equation directly for each of a sequence of finite-horizon problems. The value of the policy $\pi(s)$ is calculated within $V(s)$ whenever it is needed, getting the Bellman optimality equation. We get the combined step

$$V_{n+1}(s) = \max_{a\in\mathcal{A}}\{R(s,a) + \gamma \sum_{s^{'}\in\mathcal{S}} P(s^{'}|s,a)V_n(s^{'})\}$$

  where $n \in \mathbb{N}$ is the iteration number. Given V_0, we repeatedly compute V_{n+1} for all states s, until V converges with the left-hand side equal the right-hand side. That is, as $n \to \infty$, then $|V_n - V_{\pi*}| \to 0$ uniformly over all states. This is illustrated in Algorithm (31). Here, the values of V_n and V_{n-1} are held in two arrays, and once V_n has been computed, the values of V_{n-1} are no-longer needed, and the array used to hold them may be reused for the values of V_{n+1} as they are computed in turn. Note, provided that the values of all states are updated often enough, the value iteration method also converges if the values of individual states are updated in an arbitrary order.

- In PI, step (1) is performed once

$$V_{n+1}(s) = R(s,a) + \gamma \sum_{s^{'}\in\mathcal{S}} P(s^{'}|s,a)V_n(s^{'})$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

and then step (2) is repeated until it converges. Then, step (1) is again performed once and so on. In the case of the value function, Equations (15.3.16) and (15.3.17) are used in step (2). Instead of repeating step (2) for convergence, it may be formulated and solved as a set of linear equations. These equations are merely obtained by making $s = s^{'}$ in the step (2) equation. Thus, repeating step two for convergence can be interpreted as solving the linear equations by Relaxation (iterative method). Note, the entire evaluation function has to be recalculated at each stage, which is expensive. It is illustrated for the action-value function in Algorithm (32).

Algorithm 31 Value Iteration Algorithm

Require: Initialise V_0
Require: $n = 0$
1: **repeat**
2: For each state s do

$$V_{n+1}(s) = \max_{a \in \mathcal{A}} \{R(s,a) + \gamma \sum_{s^{'} \in \mathcal{S}} P(s^{'}|s,a) V_n(s^{'})\}$$

3: $n = n + 1$
4: **until** $n > N$ or $|V_{n+1} - V_n| < \epsilon$.

Algorithm 32 Policy Improvement Algorithm

Require: Initialise the policy π
1: **repeat**
2: Compute the value function V_π
3: Compute the action-values $Q_\pi(s,a)$ for all (s,a)
4: Update the policy for all state s by

$$\pi(s) = a \text{ such that } Q_\pi(s,a) = \max_{a \in \mathcal{A}} Q_\pi(s,a)$$

5: **until** there is no change in π at step (4).

Note, the entire evaluation function needs to be recalculated at each step, which is expensive.

15.3.4.2 Unknown transition function and reward

When the state transition function P and the reward function R are not known, we must rely on experimentation alone. Both Monte Carlo algorithms and temporal differences (TD) procedures can learn directly from raw experience without a model of the environment dynamics. TD combine the sampling of Monte Carlo algorithms with the bootstrapping of DP algorithms.

15.3.4.3 Stochastic approximation

Stochastic approximation methods are a family of iterative methods used for root-finding problems, or, for optimisation problems. The recursive update rules can be used, among other things, for solving linear systems when the collected data is corrupted by noise, or for approximating extreme values of functions which cannot be computed directly, but only estimated via noisy observations.
Stochastic approximation algorithms deal with a function of the form $f(x) = E[F(x,\xi)]$, which is the expected value of a function depending on a random variable ξ. We want to find the properties of the function f without evaluating it directly. Instead, stochastic approximation algorithms use random samples of $F(x,\xi)$ to efficiently approximate

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

properties of f such as zeros or extrema. The earliest, and prototypical, algorithms of this kind are the Robbins et al. [1951] and Kiefer et al. [1952] algorithms.

15.3.4.3.1 Description We can approximate the Bellman optimality equation by replacing the expectation with its empirical average (see Appendix (24.3.1)). Robbins et al. [1951] presented a methodology for solving a root finding problem, where the function is represented as an expected value. Rather than directly summing the samples, they proposed to add data points one by one, and iteratively update the running estimate of the mean $\widehat{x}_n$, where n is the number of iteration or the number of data points in the dataset. We let $M(x)$ denote the expected value at level x of the response to a certain experiment. We want to find the solution (unique root) $x = \theta$ of the equation

$$M(x) = \alpha$$

where α is a given constant. The authors gave a method for making successive experiments at level $x_1, x_2, ...$ in such a way that x_n will tend to θ in probability. We assume that the nature of the function $M(x)$ is unknown to the experimenter. Instead, the authors supposed that to each value x corresponds a random variable $Y = Y(x)$ with distribution function $P(Y(x) \leqslant y) = H(y|\, x)$ such that

$$M(x) = \int_{-\infty}^{\infty} y dH(y|\, x)$$

is the expected value of Y for the given x. We want to estimate θ by making successive observations on Y at levels $x_1, x_2, ...$ determined sequentially based on some experiments. Let $\{a_n\}$ be a fixed sequence of positive constants such that

$$0 < \sum_{n=1}^{\infty} a_n^2 = A < \infty$$

The nonstationary Markov chain $\{x_n\}$ is defined by taking x_1 to be an arbitrary constant and getting the update equation

$$x_{n+1} = x_n + a_n\big(\alpha - y_n\big)$$

where y_n is such that

$$P(y_n \leqslant y|\, x_n) = H(y|\, x_n)$$

Given

$$b_n = E[(x_n - \theta)^2]$$

the authors found condition under which

$$\lim_{n\to\infty} b_n = 0$$

no matter what is the initial value x_1. This limit implies the convergence in probability of x_n to θ. Blum [1954] proved that the convergence is with probability one, provided that y_n is uniformly bounded, $M(x)$ is nondecreasing, $M^{'}(\theta)$ exists and is positive, and the learning rate should satisfy the conditions

$$\lim_{N\to\infty} \sum_{n=1}^{N} a_n = \infty \text{ and } \lim_{N\to\infty} \sum_{n=1}^{N} a_n^2 < \infty$$

The authors showed that under these constraints, the iterative method of computing the mean converges to a true mean with probability one. In general, the optimal choice of the learning rate α_n is problem dependent. Some examples are:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\alpha_n = \frac{A}{B+n} \text{ and } \alpha_n = \frac{\log n}{n} \text{ with } n \geqslant 2$$

15.4 Introduction to stochastic control

For more details see Kirk [1970], Fleming et al. [1975], Davis [1993], among others. We follow the notation in Oksendal [1998]. A more general description with proofs is given in Touzi [2010].

15.4.1 Description of the problem

We let $G = \mathbb{R} \times \mathbb{R}^n$ be the parabolic interior of the state space. We let the state of a system be represented by an Ito process X with dynamics

$$dX(t) = dX_u(t) = b(t, X(t), u_t)dt + \sigma(t, X(t), u_t)dW(t)$$

where $X(t) \in \mathbb{R}^n$, $b : \mathbb{R} \times \mathbb{R}^n \times U \to \mathbb{R}^n$, $\sigma : \mathbb{R} \times \mathbb{R}^n \times U \to \mathbb{R}^{n\times n}$ and W is an m-dimensional Brownian motion.

***Remark* 15.4.1** *$u_t \in U \subset \mathbb{R}^k$ is a parameter whose value can be chosen in the Borel set in order to control the process X. Thus, $u_t = u(t, \omega)$ is a stochastic process which we assume is $\mathcal{F}_t$-adapted.*

We make suitable assumptions on the functions b and σ for the SDE to admit a solution. We denote that solution $\{X_{s,x}(h); h \geqslant s\}$ and let the probability law of $X(t)$ starting at x for $t = s$ be $Q_{s,x}$ such that

$$Q_{s,x}[X(t_1) \in F_1, ..., X(t_k) \in F_k] = P_0[X_{s,x}(t_1) \in F_1, ..., X_{s,x}(t_k) \in F_k]$$

for $s \leqslant t_i$, $F_i \subset \mathbb{R}^n$, $1 \leqslant i \leqslant k$, $k = 1, 2, ...$
We let the utility function $F : \mathbb{R} \times \mathbb{R}^n \times U \to \mathbb{R}$ and the bequest function $K : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ be given continuous functions, we also let G be a fixed domain in $\mathbb{R} \times \mathbb{R}^n$ and let $\widehat{T}$ be the first exit time after s from G for the process $\{X_{s,x}(r); r \geqslant s\}$, that is,

$$\widehat{T} = \widehat{T}_{s,x}(\omega) = \inf\{r > s; (r, X_{s,x}(r, \omega)) \notin G\} \leqslant \infty$$

To simplify notation, we apply a time shift to u_t and introduce

$$Y(t) = (s + t, X_{s,x}(s + t)) \text{ for } t \geqslant 0 \text{ , } Y(0) = (s, x)$$

so that the above SDE becomes

$$dY(t) = dY_u(t) = b(Y(t), u_t)dt + \sigma(Y(t), u_t)dW(t)$$

The probability law for $Y(t)$ is $Q_{s,x} = Q_y$ and

$$\int_s^{\widehat{T}} F_{u_r}(r, X(r))dr = \int_0^T F_{u_{s+t}}(Y(t))dt$$

where

$$T = \inf\{t > 0; Y(t) \notin G\} = \widehat{T} - s$$

Further,

$$K(\widehat{T}, X(\widehat{T})) = K(Y(T))$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

For $y = (s, x)$, the performance (or gain) function is given by

$$J_u(y) = E_y[\int_0^T F_{u_t}(Y(t))dt + K(Y(T))I(T < \infty)] \tag{15.4.25}$$

where we assume

$$E_y[\int_0^T F_{u_t}(Y(t))dt + K(Y(T))I(T < \infty)] < \infty \text{ for all } y, u \tag{15.4.26}$$

and where $F_u(z) = F(z, u)$.
The stochastic control problem is, for each $y \in G$, to find the number $\Phi(y)$ and a control $u^* = u^*(t, \omega) = u^*(y, t, \omega)$ such that

$$\Phi(y) = \sup_{u(t,\omega)} J_u(y) = J_{u^*}(y)$$

where the supremum is taken over all processes $\{u_t\}$ with values in U. If it exists, u^* is called an optimal control and Φ is called the optimal performance or the value function (VF). Some examples of controls are:

- deterministic control: $u(t, \omega) = u(t)$, also called open loop control.
- closed loop or feedback control, where the process $\{u_t\}$ is $\mathcal{M}_t$-adapted, where $\mathcal{M}_t$ is the σ-algebra generated by $\{X_u(r); r \leqslant t\}$.
- Markov control: $u(t, \omega) = u_0(t, X(t, \omega))$ for some function $u_0 : \mathbb{R}^{n+1} \to U \subset \mathbb{R}^k$. Here u does not depend on the starting point $y = (s, x)$. The value we choose at time t only depends on the state of the system at this time. Also, $X(t)$ becomes an Ito diffusion, and in particular a Markov process. In that setting the control is denoted $u(Y) = u(t, X(t))$.

15.4.2 HJB: the Markov control

We want to describe the local behaviour of the value function Φ by means of the dynamic programming equation, or Hamilton-Jacobi-Bellman equation.

15.4.2.1 Description

Among the different types of control, we will only consider the Markov control. In that setting, the dynamics of the system become

$$dY(t) = b(Y(t), u(Y(t)))dt + \sigma(Y(t), u(Y(t)))dW(t)$$

For $v \in U$ and $f \in C_0^2(\mathbb{R} \times \mathbb{R}^n)$, the diffusion operator is

$$(\mathcal{L}_v f)(y) = \frac{\partial f}{\partial s}(y) + \sum_{i=1}^n b_i(y, v)\frac{\partial f}{\partial x_i}(y) + \sum_{i,j=1}^n a_{ij}(y, v)\frac{\partial^2 f}{\partial x_i \partial x_j}(y)$$

where $a_{ij} = \frac{1}{2}(\sigma\sigma^\top)_{ij}$, $y = (s, x)$ and $x = (x_1, ..., x_n)$. For each function u, the solution $Y(t) = Y_u(t)$ is an Ito diffusion with generator A given by

$$(Af)(y) = (\mathcal{L}_{u(y)} f)(y) \text{ for } f \in C_0^2(\mathbb{R} \times \mathbb{R}^n)$$

For $v \in U$, we define $F_v(y) = F(y, v)$. We get the first fundamental result in stochastic control theory.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 15.4.1 *HJB equation (I)*
Define

$$\Phi(y) = \sup\{J_u(y); u = u(Y) \text{ Markov control }\}$$

Suppose that $\Phi \in C^2(G) \cap C(\overline{G})$ is bounded, $T < \infty$ a.s. Q_y for all $y \in G$ and that an optimal Markov control u^* exists. Suppose ∂G is regular for $Y_{u^*}(t)$. Then

$$\sup_{v \in U}\{F_v(y) + (\mathcal{L}_v\Phi)(y)\} = 0 \text{ for all } y \in G$$

and

$$\Phi(y) = K(y) \text{ for all } y \in \partial G$$

The above supremum is obtained if $v = u^*(y)$ where $u^*(y)$ is optimal. That is,

$$F(y, u^*(y)) + (\mathcal{L}_{u^*(y)}\Phi)(y) = 0 \text{ for all } y \in G$$

***Remark* 15.4.2** *The HJB (I) equation states that if an optimal control u^* exists, then we know that its value v at the point y is a point v where the function*

$$v \to F_v(y) + (\mathcal{L}_v\Phi)(y)\,,\ v \in U$$

attains its maximum, and the latter is 0. Thus, the stochastic control problem is transformed into finding the maximum of a real function in $U \subset \mathbb{R}^k$.

So far HJB (I) only states that it is necessary that $v = u^*(y)$ is the maximum of this function. We also need to know if it is sufficient. In our setting we want to know if $u_0(Y)$ is an optimal control. This is the case under some conditions:

Theorem 15.4.2 *HJB (II), a converse of HJB (I)*
Let ϕ be a function in $C^2(G) \cap C(\overline{G})$ such that, for all $v \in U$,

$$F_v(y) + (\mathcal{L}_v\phi)(y) \leqslant 0\,,\ y \in G$$

with boundary values

$$\lim_{t \to T} \phi(Y(t)) = K(Y(T)) \cdot I(T < \infty) \text{ a.s. } Q_y$$

and such that $\{\phi(Y(\tau)); \tau \leqslant T\}$ is uniformly Q_y-integrable for all Markov controls u and all $y \in G$. Then

$$\phi(y) \geqslant J_u(y)$$

for all Markov controls u and all $y \in G$.
Further, if for each $y \in G$ we have found $u_0(y)$ such that

$$F_{u_0(y)} + (\mathcal{L}_{u_0(y)}\phi) = 0$$

then $u_0 = u_0(y)$ is a Markov control such that

$$\phi(y) = J_{u_0(y)}(y)$$

and hence u_0 must be an optimal control and $\phi(y) = \Phi(y)$.

While considering only Markov controls is too restrictive, if some extra conditions are satisfied, one can always obtain as good performance with the latter as with an arbitrary $\mathcal{F}_t$-adapted control.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 15.4.3 *Let*

$$\Phi_M(y) = \sup\{J_u(y); u = u(Y) \text{ Markov control }\}$$

and

$$\Phi_a(y) = \sup\{J_u(y); u = u(t,\omega)\ \mathcal{F}_t \text{ adapted control }\}$$

Suppose there exists an optimal Markov control $u_0 = u_0(Y)$ for the Markov control problem such that all the boundary points of G are regular with respect to $Y_{u_0}(t)$ and that Φ_M is a bounded function in $C^2(G) \cap C(\overline{G})$. Then

$$\Phi_M(y) = \Phi_a(y) \text{ for all } y \in G$$

15.4.2.2 Some examples

Some examples of stochastic control problems are:

- The linear stochastic regulator problem: Suppose that the state $X(t)$ of a system at time t is given by a linear SDE

$$dX(t) = \big(H(t)X(t) + M(t)u_t\big)dt + \sigma(t)dW(t)\ ,\ t \geqslant s\ ,\ X(s) = x$$

  and the cost is of the form

$$J_u(s,x) = E_{s,x}[\int_s^{t_1} \big(X^\top(t)C(t)X(t) + u_t^\top D(t)u_t\big)dt + X^\top(t_1)RX(t_1)]\ ,\ s \leqslant t_1$$

  where all the coefficients $H(t) \in \mathbb{R}^{n\times n}$, $M(t) \in \mathbb{R}^{n\times k}$, $\sigma(t) \in \mathbb{R}^{n\times m}$, $C(t) \in \mathbb{R}^{n\times n}$, $D(t) \in \mathbb{R}^{k\times k}$ and $R \in \mathbb{R}^{n\times n}$ are t-continuous and deterministic. The problem is to choose the control $u = u(t, X(t)) \in \mathbb{R}^k$ such that it minimises $J_u(s,x)$.

- An optimal portfolio selection problem: We let $X(t)$ denotes the wealth of an agent at time t. The agent has the choice between two different investments, a risky one (a stock) with dynamics

$$\frac{dS_1(t)}{S_1(t)} = bdt + \sigma dW_t$$

  and a risk-free one with dynamics

$$\frac{dS_0(t)}{S_0(t)} = rdt$$

  The wealth at time t is $X(t) = \theta_0 S_0(t) + \theta_1 S_1(t)$, where θ_i is the quantity of asset S_i. At each time, the agent can choose the fraction $u = \frac{\theta_1 S_1(t)}{X(t)}$ of his wealth he will invest in the risky asset, thus investing $(1-u)$ in the risk-free asset. Given the dynamics for the risky asset, we obtain the SDE for the wealth $X(t) = X_u(t)$ as

$$dX(t) = X(t)\big(bu + r(1-u)\big)dt + \sigma u X(t)dW_t$$

  Starting with the wealth $X(t) = x > 0$ at time t, allowing for no borrowing, the agent wants to maximise the expected utility of the wealth at some future time $t_1 > t$. Letting $N : [0,\infty) \to [0,\infty)$, $N(0) = 0$, be a utility function, the problem is to find the value function $\Phi(s,x)$ and a Markov control $u^* = u^*(t, X(t))$, $0 \leqslant u^* \leqslant 1$, such that

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\Phi(s,x) = \sup\{J_u(s,x); u \text{ Markov control } , 0 \leqslant u \leqslant 1\} = J_{u^*}(s,x)$$

where $J_u(s,x) = E_{s,x}[N(X_u(T))]$ where T is the first exit time from the region $G = \{(r,z); r < t_1 , z > 0\}$. Details are given in Appendix (28.3).

15.4.3 Terminal conditions

We now consider the case of constraints on Markov controls $Y_u(t)$, such as terminal condition at time $t = T$. One way of solving this problem is to apply a sort of Lagrange multiplier. The problem is as follows: Find $\Phi(y)$ and $u^*(y)$ such that

$$\Phi(y) = \sup_{u \in \mathcal{K}} J_u(y)$$

where

$$J_u(y) = E^y[\int_0^T F_u(Y_u(t))dt + K(Y_u(T))]$$

and where the supremum is taken over the space $\mathcal{K}$ of all Markov controls $u : \mathbb{R}^{n+1} \to U \subset \mathbb{R}^k$ such that

$$E^y[M_i(Y_u(T))] = 0 \, , i = 1, 2, ..., l$$

where $M = (M_1, ..., M_l) : \mathbb{R}^{n+1} \to \mathbb{R}^l$ is a given continuous function, and

$$E^y[|M(Y_u(T))|] < \infty \text{ for all } y, u$$

We can obtain a related, but unconstrained problem: For each $\lambda \in \mathbb{R}^l$ and each Markov control u, we define

$$J_{u,\lambda}(y) = E^y[\int_0^T F_u(Y_u(t))dt + K(Y_u(T)) + \lambda \cdot M(Y_u(T))]$$

The problem is to find $\Phi_\lambda(y)$ and $u^*_\lambda(y)$ such that

$$\Phi_\lambda(y) = \sup_u J_{u,\lambda}(y) = J_{u^*_\lambda,\lambda}(y)$$

without terminal conditions.

Theorem 15.4.4 *Suppose that for all $\lambda \in \Lambda \subset \mathbb{R}^l$ we can find $\Phi_\lambda(y)$ and u^*_λ solving the above unconstrained stochastic control problem. Further, suppose that there exists $\lambda_0 \in \Lambda$ such that*

$$E^y[M(Y_{u^*_{\lambda_0}}(T))] = 0$$

Then $\Phi(y) : \Phi_{\lambda_0}(y)$ and $u^* := u^*_{\lambda_0}$ solves the original constrained stochastic control problem.

15.4.4 Dynamic programming

We slightly modify the notation in Section (15.4.1) and follow the description given by Touzi [2010].

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

15.4.4.1 Dynamic programming principle

We let $G = [0, T) \times \mathbb{R}^d$, where $T \in [0, \infty]$, be the parabolic interior of the state space. We denote $\overline{G} = \text{cl}(G)$ its closure. We let the state of a system be represented by an Ito process X with dynamics

$$dX(t) = dX_u(t) = b(t, X(t), u_t)dt + \sigma(t, X(t), u_t)dW(t)$$

where $X(t) \in \mathbb{R}^d$, $b : G \times U \to \mathbb{R}^d$, $\sigma : G \times U \to \mathbb{R}^{n\times d}$ and W is an n-dimensional Brownian motion. We let

$$f, k : G \times U \to \mathbb{R} \text{ and } g : \mathbb{R}^d \to \mathbb{R}$$

be given functions. We assume f, k are continuous and $\max(-k, 0)$ is uniformly bounded. Further, f and g satisfy the quadratic growth condition.
We consider the value function on $G \times U$ given by

$$J(t, x, u) = E[\int_t^T \beta_u(t, s)f(s, X_{t,x,u}(s), u_s)ds + \beta_u(t, T)g(X_{t,x,u}(T))I(T < \infty)]$$

where $\beta_u(t, s) = e^{-\int_t^s k(r,X_{t,x,u}(r),u_r)}$. We denote

$$\Phi_*(t, x) = \lim_{(t',x')\to(t,x)} \inf \Phi(t', x') \text{ and } \Phi^*(t, x) = \lim_{(t',x')\to(t,x)} \sup \Phi(t', x')$$

for all $(t, x) \in \overline{G}$.

Theorem 15.4.5 *Assume that Φ is locally bounded. Let $(t, x) \in G$ be fixed. Let $\{T_u; u \in U_t\}$ be a family of finite stopping times independent of $\mathcal{F}_t$ with values in $[t, T]$. Then*

$$\Phi(t, x) \leqslant \sup_{u\in U_t} E[\int_t^{T_u} \beta_u(t, s)f(s, X_{t,x,u}(s), u_s)ds + \beta_u(t, T_u)\Phi_*(T_u, X_{t,x,u}(T_u))]$$

Assume further that g is lower-semicontinuous and $X_{t,x,u}I_{[t,T_u]}$ is L_∞-bounded for all $u \in U_t$. Then

$$\Phi(t, x) \geqslant \sup_{u\in U_t} E[\int_t^{T_u} \beta_u(t, s)f(s, X_{t,x,u}(s), u_s)ds + \beta_u(t, T_u)\Phi_*(T_u, X_{t,x,u}(T_u))]$$

If Φ is continuous, then $\Phi = \Phi_* = \Phi^*$ and the above weak dynamic programming principle reduces to the classical dynamic principle

$$\Phi(t, x) = \sup_{u\in U} E_{t,x}[\int_t^T \beta(t, s)f(s, X(s), u_s)ds + \beta(t, T)\Phi(T, X(T))]$$

In the discrete-time framework, the dynamic programming principle becomes

$$\Phi(t, x) = \sup_{u\in U} E_{t,x}[f(t, X(t), u) + e^{-k(t+1,X(t+1),u)}\Phi(t + 1, X(t + 1))]$$

15.4.4.2 Dynamic programming equation

We are now going to derive the dynamic programming equation, or HJB, under strong smoothness assumption on the value function.
We let G_d be the set of all $d \times d$ symmetric matrices with real coefficients, and define the map $H : G \times \mathbb{R} \times \mathbb{R}^d \times M_d$ by

$$H(t, x, r, p, \gamma) = \sup_{u\in U}\{-k(t, x, u)r + b(t, x, u) \cdot p + \frac{1}{2}Tr[\sigma\sigma^\top(t, x, u)\gamma] + f(t, x, u)\}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We introduce the linear second order operator $\mathcal{L}_u$ associated to the control process $\{\beta_u(0,t)X_u(t); t \geqslant 0\}$ controlled by the constant control process u

$$\mathcal{L}_u\varphi(t,x) = -k(t,x,u)\varphi(t,x) + b(t,x,u)\cdot D\varphi(t,x) + \frac{1}{2}Tr[\sigma\sigma^\top(t,x,u)D^2\varphi(t,x)]$$

where D and D^2 denote respectively the gradient and the Hessian operators with respect to the variable x. Applying Ito's formula, we get

$$\begin{aligned}\beta_u(0,s)\varphi(s,X_u(s)) - \beta_u(0,t)\varphi(t,X_u(t)) &= \int_t^s \beta_u(0,r)\big(\partial_t + \mathcal{L}_{u_r}\big)\varphi(r,X_u(r))dr \\ &+ \int_t^s \beta_u(0,r)D\varphi(r,X_u(r))\cdot\sigma(r,X_u(r),u_r)dW_r\end{aligned}$$

for every $s \geqslant t$ and smooth function $\varphi \in C^{1,2}([t,s],\mathbb{R}^d)$ and each admissible control process u.

***Proposition* 9** *Assume the value function $\Phi \in C^{1,2}([0,T],\mathbb{R}^d)$, and let the coefficients $k(\cdot,\cdot,u)$ and $f(\cdot,\cdot,u)$ be continuous in (t,x) for all fixed $u \in U$. Then for all $(t,x) \in G$*

$$-\partial_t\Phi(t,x) - H(t,x,\Phi(t,x),D\Phi(t,x),D^2\Phi(t,x)) \geqslant 0$$

***Proposition* 10** *Assume the value function $\Phi \in C^{1,2}([0,T],\mathbb{R}^d)$ and $H(\cdot,\Phi,D\Phi,D^2\Phi) > -\infty$. Assume further that k is bounded and the function H is continuous. Then for all $(t,x) \in G$*

$$-\partial_t\Phi(t,x) - H(t,x,\Phi(t,x),D\Phi(t,x),D^2\Phi(t,x)) \leqslant 0$$

As a result of the above two propositions, we get the following theorem:

Theorem 15.4.6 *Let the conditions of the above two propositions hold. Then, the value function Φ solves the HJB equation*

$$-\partial_t - H(\cdot,\cdot,\Phi,D\Phi,D^2\Phi) = 0 \text{ on } G$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 16

Solving conditional expectations

16.1 Temporal difference learning

16.1.1 Introduction

16.1.1.1 Overview

As discussed in Section (15.1), reinforcement learning (RL) refers to the problem of computing near optimal policies in Markov Decision Processes (MDP) when the underlying model is unknown or too complex. We saw in Section (15.3.4) that RL was composed of two problems:

1. Policy Evaluation (or prediction problem), which estimates the value function V_π for a given policy π.
2. Control Problem, which finds the optimal policy.

Temporal differences (TD) focus on estimating the value function of a given feedback policy in an MDP. It originates from credit assignment problems. In a prediction-learning problem [1], a learning agent performs a sequence of actions, obtaining some rewards or penalties (punishments). Credit assignment problems occur when a sequence of decisions are made before an evaluation is received. The problem consists in apportioning the final evaluation among the preceding decisions (see Minsky [1963]). There are two types of credit assignment problems:

1. Temporal credit assignment: A learning system produces a sequence of actions before an evaluation becomes available. For instance, the sequence of moves from the start of a chess game to the final win.
2. Structural credit assignment: Assigning credit/blame to each decision in the sequence.

The temporal credit assignment problem consists in assigning credit or blame to the situations and actions involved. However, it is generally difficult to perform when the reinforcement delay is long and the environment is stochastic. The most important learning paradigm has been supervised learning paradigm (SLP) (see Section (5.1)), where the learner needs to associate pairs of items. When presented with just the first item, the learner recalls the second one. Any prediction problem can be cast in a SLP by taking the first item to be the input of a prediction function, and the second item to be the actual outcome. That is, $y = f(x)$ for the pair (x, y). However, Sutton [1988] argued that these methods are inadequate and that temporal difference (TD) are far preferable.
Nowadays, the most popular procedure for temporal credit assignment is temporal difference (TD). While conventional prediction-learning methods, such as supervised learning methods (see Section (5.1)), are driven by the error between predicted and actual outcomes, TD methods are driven by the error between temporally successive predictions. Several

[1] Using past experience with an incompletely known system to predict its future behaviour.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

TD algorithms have been proposed (see Samuel [1959], Hampson [1983], Barto et al. [1983], Holland [1986]). Sutton [1988] proved their convergence and optimality for special cases and related them to supervised learning methods.
We can distinguish between two kinds of prediction-learning problem:

1. Single-step: all information about the correctness of each prediction is revealed at once. It is a single point forecast. For example, predict next day stock prices based on today's prices.

2. Multi-step: correctness is not revealed until more than one step after the prediction is made. Only partial information relevant to its correctness is revealed at each step. For example, predict X-days ahead weather forecast.

In type $[1]$, the data comes in observation-outcome pairs, which is ideally suited for pairwise supervised learning approach. TD methods are more concerned with type $[2]$ prediction problems since type $[1]$ is a special case. Nonetheless, many single-step prediction problems are more naturally viewed as multi-step problems. This is because patterns are not unrelated, but are part of a series of related patterns, all providing information about the same classification. Sutton [1988] showed that TD methods were more efficient than supervised learning methods when the data sequences were generated by a dynamical system (see Appendix (23.4)).
TD methods were originally used to predict the probability of winning a game for each possible board situation. As an example, we consider the learning problem of predicting the wining probability of a tic-tac-toe by collecting all the training instances. Given the training instance in Figure (16.1), suppose that the estimated winning probabilities from states A, B and C are 0.8, 0.3, and 0.6, respectively. The player started at state A moved to B and then to C, and won the game.

Figure 16.1: A sequence of state transitions with estimated probabilities in a game play.

In that setting, the delta rule (see Section (5.1.4)) would increase the WP (see Equation (5.1.7)) from states A, B, and C by $(1-0.8)\alpha$, $(1-0.3)\alpha$, and $(1-0.6)\alpha$, where α is a learning rate. Here, the learning changes are proportional to the differences between the current predictions and the actual outcome (in this case 1). Storing the WPs in a lookup table, and sufficient training data, the delta rule would eventually converge to the true WP from every state.
On the other hand, $TD(0)$ minimises the error (TD error) between successive predictions. Let Δ_s be the learning change to the WP from a state s. $TD(0)$ makes the following learning changes:

$$\Delta_A = (0.3-0.8)\alpha \,,\, \Delta_B = (0.6-0.3)\alpha \,,\, \Delta_C = (1.0-0.6)\alpha$$

Sutton [1988] made three assumptions to prove convergence of $TD(0)$ to the optimal estimates:

1. sufficient training data

2. repeated presentations of training data

3. the target system is linear

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The author also showed that on multi-step prediction problems outperformed supervised learning. The family of TD methods is denoted $TD(\lambda)$, where $0 \leqslant \lambda \leqslant 1$ is called the recency factor. Then, $TD(0)$ is the difference between two successive predictions, and $TD(\lambda)$ is the sum of $TD(0)$ errors exponentially weighted by λ. In our previous example, we get the following learning

$$\begin{aligned}
\Delta_A &= \big[(0.3-0.8)+(0.6-0.3)\lambda+(1.0-0.6)\lambda^2\big]\alpha \\
\Delta_b &= \big[(0.6-0.3)+(1.0-0.6)\lambda\big]\alpha \\
\Delta_C &= (1.0-0.6)\alpha
\end{aligned}$$

Dayan [1992] and Bertsekas et al. [1996] proved asymptotic convergence of $TD(\lambda)$. Note, $TD(1)$ makes the same learning changes as the supervised learning method. As λ approaches 1, $TD(\lambda)$ behaves more like supervised learning. Further, $TD(\lambda)$ not only males prediction changes based on information from the immediately successive state, but also information from all the future states. Thus, $TD(\lambda)$ results in faster learning than $TD(0)$ at the early learning stage.

16.1.1.2 Model assumptions

As discussed in Section (16.1.3.4), the data sequences are generated by absorbing Markov processes, where the next state depends only on the current state. They are defined by a set of terminal states T, a set of nonterminal states N, and a set of transition probabilities p_{ij}, $i \in N, j \in N \cup T$, where each p_{ij} is the probability of a transition from state i to state j, given that the process is in state i. Since it is absorbing, all sequences eventually terminate. Given an initial state q_1, the absorbing Markov process generates the sequence $q_1, q_2, ..., q_{m+1}$, where $q_{m+1} \in T$. The learning algorithms do not know about the state sequence, but only of a related observation-outcome sequence $x_1, x_2, ..., x_m, z$, where each x_t is the observation vector at time t and z is the outcome of the sequence. The vector x_t is assumed to be real-valued measurements or features, and z is a real-valued scalar.
When the observation sequence is made of categorical data, we need to perform the one-hot encoding scheme to transform the data into numerical values (see Section (3.1.1.2)). For each nonterminal state i, we associate an observation vector X_i. It is assumed that if the sequence is in state i at time t, then $x_t = X_i$. Thus, even though x_t is a vector, the observation sequence represent a one-dimensional state process. That is, we only consider a 1D Markov chain. If we want to apply temporal differences to an N-dimensional state process we need to modify the theory to introduce higher-dimensional Markov chains (see Appendix (25.2.2)).

16.1.2 Temporal credit assignment

The ideal reinforcement signal provides the highest quality evaluation of the behaviour of a learning system. Temporal credit assignment algorithms can be viewed as attempts to approximate that ideal signal. The AHC-algorithm by Sutton [1984] is an example of a model developed on the basis the ideal reinforcement signal.
We give a brief overview of the algorithms for temporal credit assignment tested by Sutton [1984], and describe the Adaptive Heuristic Critic (AHC) model.

16.1.2.1 Nonassociative task

In a nonassociative task, the learning system only receives a scalar primary reinforcement signal. There is no non-reinforcement stimuli with which to associate actions. The learning system must find and consistently choose an action maximising the expected value of the primary reinforcement signal. There are two types of nonassociative reinforcement learning problems:

1. Function optimisation: They consider continuous action spaces, such as the parameter space of a control system. This is problems with continuous-valued performance measures.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. Learning automata theory (LAT) and mathematical learning theory (MLT): They consider problems with discrete action spaces, where the system chooses one out two, or one out of n, actions. Problems have discrete outcomes, such as binary-reinforcement or success/failure problems. These problems have been studied as the n-armed bandit problems.

In the case [2], algorithms are global-search algorithms that can not be trapped on local optima. When comparing the current reinforcement level with past ones, the algorithms are called reinforcement-comparison (RC) algorithms. In general, case [2] are not RC-algorithms. This is related to the independence-of-path assumption, which states that the only memory of past experience a learning system retains is contained in the probabilities of selecting each action. A task is called an unbalanced-reinforcement (UR) task if its primary reinforcement signal is biased towards a predominance of either positive values (assign credit to the system) or negative values (blame). RC algorithms provide an advantage on UR-tasks. They can produce a new (heuristic) reinforcement signal which is balanced or nearly balanced, helping learning. By definition, it retains memory about past levels of reinforcement and uses it to guide changes in its action probabilities. In a current state, in addition to knowing the action probabilities, it includes memory of past reinforcement levels. This hypothesis was tested by the author.
The author designed experiments to compare convergence rates of several algorithms. He described a series of computer simulations of ten learning algorithms, with and without RC-algorithms, and applied six nonassociative reinforcement learning tasks (binary and continuous tasks). We denote the action at time t as a_t, and r_{t+1} is the reinforcement received at time $t+1$. In a binary task, if the outcome is a success the value is $+1$ and if it is a failure it is -1. In a continuous task, the outcome is a real-valued reinforcement. For simplicity, the tasks were constructed such that action 1 is the better action on all tasks. π_t denotes the probability with which action 1 is selected at time t. Some examples of update rules are:

$$\begin{aligned} w_{t+1} &= w_t + \alpha r_{t+1}\big(a_t - \pi_t\big) \\ w_{t+1} &= w_t + \alpha\big(r_{t+1} - r_t\big)\big(a_t - \pi_t\big) \\ w_{t+1} &= w_t + \alpha\big(r_{t+1} - P_t\big)\big(a_t - \pi_t\big) \end{aligned}$$

where $w_0 = 0$, $\pi_0 = \frac{1}{2}$, with learning rate $\alpha > 0$. Further, the expected level of reinforcement P_t at time t, is updated as follows:

$$P_{t+1} = P_t + \beta\big(r_{t+1} - P_t\big)$$

where $P_0 = r_1$ and $\beta = 0.2$. While some algorithms directly updated the probability π_t, others updated a modifiable parameter w_t that determines the probability based on

$$\pi_t = \Phi(\frac{w_t}{\sigma_a})$$

where Φ is the unit normal distribution function, and standard deviation $\sigma_a = 0.3$.
In RC-algorithms, the author compared the current reinforcement with the immediately preceding one (short-term measure), getting

$$r_{t+1} - r_t$$

These algorithms performed much better across tasks than the ones with no RC. They improve performance on UR-tasks.
We can also compare the current reinforcement with a long-term measure of prior reinforcement. This is a real-valued variable called the expected level of reinforcement, denoted P_t at time t. Here, w is updated according to

$$\sigma_{t+1} = r_{t+1} - P_t \tag{16.1.1}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where P_t is an exponentially-weighted average, or trace, of the preceding reinforcement values $r_t, r_{t-1}, ...,$ with greater weight going to more recent reinforcement values. Here, σ_{t+1} is the error in the predicted reinforcement. The updated rule is given above. It tracks the level of r_{t+1} with learning rate β. In the case $\beta = 0.2$, it tracks r_{t+1} relatively slowly. These types of algorithms performed significantly better than those with short-term measure.

16.1.2.2 Associative task

In associative reinforcement-learning (ARL) tasks, a mapping from input to output must be formed by the learning system. Stimuli and actions must be associated so that the occurrence of a stimulus evokes the most suitable or appropriate action. For instance, in learning in pattern classification, associations are learned between patterns (stimuli) and classifications (actions). But, this is not reinforcement learning because there is no search for the best classifications. It is provided by a teacher during the training sequence. The author compared the performance of eleven associative-earning algorithms on twelve ARL-tasks. The algorithms are generalisation of the ones discussed in the previous section. The author consider three approaches to the problem:

1. The independent-associations
2. The stimulus-sampling-theory
3. The linear-mapping approach

Generalisation between stimuli occurs when training to one stimulus transfers and influences behaviour (action selections) to another stimulus. When the action learned in the first stimulus is more likely to occur in the second one after training, the transfer is positive, and the two stimuli are similar.
Some example of updating rules are:

$$\begin{aligned} w_{i,t+1} &= w_{i,t} + \alpha r_{t+1}\big(a_t - \pi_t\big)x_{i,t} \\ w_{i,t+1} &= w_{i,t} + \alpha\big(r_{t+1} - r_t\big)\big(a_t - \pi_t\big)x_{i,t} \\ w_{i,t+1} &= w_{i,t} + \alpha\big(r_{t+1} - P_t\big)\big(a_t - \pi_t\big)x_{i,t} \end{aligned}$$

where $w_{i,0} = 0$, $v_{i,0} = 0$, $\pi_0 = \frac{1}{2}$, with learning rate $\alpha > 0$. Further,

$$P_t = \sum_{i=1}^{n} v_{i,t}x_{i,t} \text{ and } S_t = \sum_{i=1}^{n} w_{i,t}x_{i,t}$$

where x_i is a stimulus component. While some algorithms directly updated the probability π_t that $a_t = 1$ given x_t. In some cases, determining a_t implies that

$$\pi_t = \Phi(\frac{S_t}{\sigma_a})$$

where Φ is the unit normal distribution function. In that setting, $a_t = 1$ if $S_t + \eta_t > 0$, and 0 otherwise. Here, η_t is normally distributed with mean 0 and standard deviation $\sigma_a = 0.3$.
Algorithms comparing the current reinforcement with the immediately preceding one are not successful. This is because for many associative tasks, it is possible to obtain high reinforcement levels in the presence of one stimulus and only low ones in the presence of another stimulus. So, we should compare reinforcement received on one step with that from the previous step only if the same stimulus occurred. We need to understand how past reinforcement levels for different stimuli are recorded and kept separate from each other. One solution is to view the sequence of stimuli, and the reinforcement levels obtained by acting in response to them, as a training sequence for supervised learning pattern classification. This is the Widrow-Hoff algorithm (see Widrow et al. [1960]). We need a modifiable parameter vector v_t in addition to the vector w_t. Here,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- w_t maps a stimulus to an action. So, w is called the action-association vector.
- v_t maps a stimulus to an average of the reinforcement levels obtained when the stimulus occurred in the past. So, v is called the reinforcement-association vector.

The average of past reinforcement levels is called the predicted reinforcement, and denoted $P_t \in \mathbb{R}$. It is computed above. v_t is updated as follows:

$$v_{i,t+1} = v_{i,t} + \beta\big(r_{t+1} - P_t\big)x_{i,t}$$

for $t = 0, 1, ..., i = 1, ..., n$, $v_{i,0} = 0$, and $\beta = 0.1$. Similarly to Equation (16.1.1), the error in the predicted reinforcement is given by

$$r_{t+1} - P_t$$

It is also the quantity needed as a comparison of the current reinforcement level with past reinforcement levels associated with the current stimulus. When the current reinforcement r_{t+1} is greater (less) than P_t, w_i is changed so as to make the action selected more (less) likely as a response to the stimulus. Note, RC-algorithms have similar advantage in associative-learning tasks as in the previous section.

16.1.2.3 Delayed reinforcement

So far, we have considered tasks where actions only affect the reinforcement received on the following step. They are called immediate-reinforcement tasks. Tasks where some of the effect of an action on reinforcement are delayed by several time steps are called delayed-reinforcement tasks. In that case, r_{t+1} may depend on $a_t, a_{t-1}, ...,$. The main point of secondary reinforcement mechanism is to reduce the delays between actions and reinforcement. Secondary reinforcement is used when actions of the learning system affect subsequent stimuli. In that setting, algorithms associate stimuli with forthcoming reinforcement and then use the information provided by their occurrence to better assign temporal credit. Such algorithms were provided by Samuel [1959] and Witten [1977]. Sutton [1984] presented the Adaptive Heuristic Critic (AHC) algorithm, which we are now going to describe.

16.1.2.3.1 Primary reinforcement In general, we assign credit according to an exponentially decreasing function λ^k of the number k of time steps elapsing between action and reinforcement. We can combine the frequency heuristic with an exponential decaying recency heuristic, getting the update rule for the action-association vector as follows:

$$w_{i,t+1} = w_{i,t} + \alpha\sigma_{t+1}(1-\lambda)\sum_{k=0}^{t-1}\lambda^k e_{i,t-k}$$

for $t > 0$ and $i = 1, ..., n$, where $w_t = (w_{1,t}, ..., w_{n,t})$ is the vector of action-association weights, $e_{i,t}$ is the eligibility of its ith component (see Klopf [1972]), α is a positive learning rate, and σ_{t+1} is a heuristic reinforcement. The product $\sigma_{t+1}\lambda^k e_{i,t-k}$ is the credit due to the behaviour at $t-k$ that should be assigned to w_i given σ_{t+1}. The main advantage of the exponential decay form of the recency heuristic is that the sum in the above equation can be computed iteratively. For ant time sequence e_i (to be defined), we let $\overline{e}_i$ be the time sequence resulting from the application of a trace operator to e_i. That is,

$$\overline{e}_{i,t} = (1-\lambda)\sum_{k=0}^{t-1}\lambda^k e_{i,t-k}\,,\, t > 0\,,\, \overline{e}_{i,0} = 0 \tag{16.1.2}$$

Then, we get

$$\overline{e}_{i,t} = (1-\lambda)e_{i,t} + \lambda\overline{e}_{i,t-1}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

When $\lambda = 0$, then $\overline{e}_i$ simplifies to e_i. It requires only a single memory variable per component of w, and remember all past behaviour. It is a standard discrete-time recursive linear filter. The update rule in the above equation can be rewritten as

$$w_{i,t+1} = w_{i,t} + \alpha\sigma_{t+1}\overline{e}_{i,t} \tag{16.1.3}$$

From the RC-algorithms of the previous sections (see Equation (16.1.1)), we can define the heuristic reinforcement signal as

$$\sigma_{t+1} = r_{t+1} - P_t$$

where r is the primary reinforcement and P_t is the predicted reinforcement computed as in the previous section. Some update rules are:

$$\begin{aligned} w_{i,t+1} &= w_{i,t} + \alpha\big(r_{t+1} - P_t\big)\overline{e}_{i,t} \\ v_{i,t+1} &= v_{i,t} + \beta\big(r_{t+1} - P_t\big)\overline{x}_{i,t} \end{aligned}$$

and

$$\begin{aligned} w_{i,t+1} &= w_{i,t} + \alpha\big(r_{t+1} - \overline{P}_t\big)\overline{e}_{i,t} \\ v_{i,t+1} &= v_{i,t} + \beta\big(r_{t+1} - \overline{P}_t\big)\overline{x}_{i,t} \end{aligned}$$

Note, the Widrow-Hoff rule (see Section (5.1.4)) gives

$$v_{i,t+1} = v_{i,t} + \beta\big(r_{t+1} - P_t\big)x_{i,t}$$

Since P_t is an estimate of r_{t+1}, then $\sigma_{t+1} = r_{t+1} - P_t$ is an error term. It can be generalised to delayed reinforcement by correlating the error σ with all preceding stimuli as follows:

$$\begin{aligned} v_{i,t+1} &= v_{i,t} + \beta\sigma_{t+1}(1-\lambda)\sum_{k=0}^{t-1}\lambda^k x_{i,t-k} \\ &= v_{i,t} + \beta\sigma_{t+1}\overline{x}_{i,t} \end{aligned}$$

and $\overline{x}_{i,t}$ is computed with the trace operator in Equation (16.1.2) applied to x_i.
In the last algorithm, we assume that the prediction of reinforcement at time $t+1$ should depend on all past stimuli and not just the stimulus at time t. It is denoted by $\overline{P}_t$ and the error is $\sigma_{t+1} = r_{t+1} - \overline{P}_t$.
Adding exponentially-decaying eligibility traces to the algorithms of the previous section greatly improved their performance on delayed-reinforcement tasks. However, since we do not know the length of the delay, there is a trade-off between performance on short-delay tasks and the one on long-delay tasks.

16.1.2.3.2 Neuronlike adaptive elements Barto et al. [1983] proposed a system made of two neuronlike adaptive elements to solve difficult learning control problem. It consists of a single associative search element (ASE) and a single adaptive critic element (ACE). Applied to the task of balancing a pole, the ASE constructs associations between input and output by searching under the influence of reinforcement feedback, and the ACE consists of a more informative evaluation function than the feedback alone. The ACE is a refinement of the model of classical conditioning proposed by Sutton et al. [1981]. It is consistent with the Rescorla-Wagner model of classical conditioning (see Appendix (16.5)). In Figure (16.2), the ASE and ACE are configured for pole balancing task.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 16.2: ASE and ACE configured for pole balancing task.

In the ASE there are n pathways for reinforcement input, and a single output pathway. We let $x_{i,t}$, $1 \leqslant i \leqslant n$, denote the real-valued signal on the ith reinforcement input pathway at time t, and y_t denote the output at time t. A real-valued weight $w_{i,t}$ is associated to each input pathway i. Given the input vector $X_t = (x_{1,t}, ..., x_{n,t})$ the output is defined as

$$y_t = f\Big(\sum_{i=1}^{n} w_{i,t} x_{i,t} + \epsilon_t\Big)$$

where ϵ_t is some noise at time t. It is a real random variable with density function d. In general, it is a mean zero Gaussian distribution with variance σ^2. f is the function

$$f(x) = \begin{cases} +1 \text{ if } x \geqslant 0 \\ -1 \text{ if } x < 0 \end{cases}$$

The weights w_i, $1 \leqslant i \leqslant n$, change over discrete time as

$$w_{i,t+1} = w_{i,t} + \alpha r_t e_{i,t}$$

where α is the learning rate, r_t is the reinforcement at time t, and $e_{i,t}$ is the eligibility at time t of the input pathway i. When some conditions hold for the input pathway i, then the latter becomes eligible to have its weight modified, and it remains eligible for some period of time after the conditions cease to hold. Thus, w_i changes based on the reinforcement received during periods of eligibility. When reinforcement indicates improved performance, the weights of the eligible pathways are modified to make the element more likely to improve performance. Here, $e_{i,t}$ is a trace of the product $y_\tau x_{i,\tau}$ for times τ preceding t. The authors considered the exponentially decaying eligibility traces e_i using the linear difference equation

$$e_{i,t+1} = \delta e_{i,t} + (1 - \delta) y_t x_{i,t}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where δ, $0 \leqslant \delta < 1$, determines the trace decay rate. In the boxes local-demon algorithm, the factor $x_{i,t}$ triggers the eligibility trace (a count), whenever box i is entered.
In the example of the boxes system, the ACE stores in each box a prediction (expectation) of the reinforcement that can be obtained from the environment by choosing an action for that box. This prediction is used to determine a reinforcement signal that it delivers to the ASE whenever the box is entered by the cart-pole state. Here, the idea is that the predictions predict the reinforcement, but also future predictions of reinforcement. Similarly to the ASE, the ACE has n pathways for reinforcement input and a single output. σ_t denotes the real-valued output signal at time t. It is the improved reinforcement signal used by the ASE in place of r. $v_{i,t}$ is the weight of the ith pathway. We let P_t be the prediction of eventual reinforcement. It is a function of the input vector X_t given by

$$P_t = \sum_{i=1}^{n} v_{i,t} x_{i,t}$$

For P_t to converge to an accurate prediction, the update rule for v_i is

$$v_{i,t+1} = v_{i,t} + \beta \sigma_t \overline{x}_{i,t}$$

where β is the rate of change of v_i and $\overline{x}_{i,t}$ is the value at time t of a trace of the input variable x_i, and

$$\sigma_t = r_t + \gamma P_t - P_{t-1}$$

where γ, $0 < \gamma \leqslant 1$, is the discount factor.

***Remark* 16.1.1** *σ_t is to be compared with Equation (16.1.1). We see that the reinforcement r_{t+1} is approximated by $r_t + \gamma P_t$.*

The trace $\overline{x}_{i,t}$ acts similarly to the eligibility trace defined above. However, an input pathway gains positive eligibility whenever a nonzero signal is present on that pathway, irrespective of the action. It is calculated by using the linear difference equation

$$\overline{x}_{i,t+1} = \lambda \overline{x}_{i,t} + (1 - \lambda) x_{i,t}$$

where λ, $0 \leqslant \lambda < 1$ is the learning rate of the trace. Thus, the weight changes whenever the actual reinforcement r_t plus the current prediction P_t differs from the value P_{t-1} that was predicted for this sum. It is related to the delta rule and provides a way of finding weight values such that P_{t-1} approximates $r_t + \gamma P_t$. That is, the learning rule attempts to predict its own prediction, producing predictions that tend to be the earliest possible indications of eventual reinforcement. This is at the heart of temporal differences (TD). The output of the ACE, σ_t, is the improved (or internal) reinforcement signal. Replacing r with σ in the ASE, the weight updating rules for ASE and ACE only differ in their forms of eligibility traces. The traces in ASE are conditional on its output, which is not the case in the ACE.

16.1.2.4 The AHC-algorithm

The Adaptive Heuristic Critic (AHC) algorithm, introduced by Sutton [1984], is closely related to the model of classical conditioning by Sutton et al. [1981] and Barto et al. [1982], and influenced by the work of Klopf [1972] [1982]. All these models are related to each other via the concept of an ideal reinforcement signal. It is positive whenever the immediately preceding action is better than average for the situation in which it was taken, and negative when worse than average. Thus, it indicates the value of the selected action relative to the average of the values of all possible actions, each weighted by the probability of being chosen in the given situation. It should stimulate the maximisation of cumulative performance.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

16.1.2.4.1 Description In the AHC-algorithm, a linear mapping is used to associate predictions of forthcoming reinforcement with stimuli. The stimuli at time t are given by the vector x_t, where the component x_i, $1 \leqslant i \leqslant n$, is associated with a memory variable v_i indicating the extent to which the presence of the stimulus is linked to high or low forthcoming reinforcement. The prediction of reinforcement forthcoming after time t, made by the learning system at time s (using v_s), is denoted by $P_{s,t}$ and defined as

$$P_{s,t} = \sum_{i=1}^{n} v_{i,s} x_{i,t} \tag{16.1.4}$$

We want that $P_{s,t}$ becomes a good estimate of forthcoming reinforcement. Thus, we need a good heuristic reinforcement signal as a basis for correcting estimates of $P_{s,t}$ by changing v. This is obtained with the following update rule:

$$v_{i,t+1} = v_{i,t} + \beta \sigma_{t+1} \overline{x}_{i,t} \, , v_{i,0} = 0 \, , 1 \leqslant i \leqslant n \, , t = 0, 1, ... \tag{16.1.5}$$

where $\overline{x}_{i,t}$ is the trace in Equation (16.1.2) applied to x_i. Following the neuronlike adaptive elements in the above section, the heuristic reinforcement signal in the AHC algorithm is given by

$$\sigma_{t+1} = r_{t+1} + \gamma P_{t,t+1} - P_{t,t} \tag{16.1.6}$$

where γ, $0 \leqslant \gamma \leqslant 1$, is a scalar discount rate parameter.

***Remark* 16.1.2** *For both $P_{t,t+1}$ and $P_{t,t}$, the reinforcement-association vector is v_t, seen at time t, but $P_{t,t}$ is the prediction of reinforcement forthcoming after time t and $P_{t,t+1}$ is the prediction of reinforcement forthcoming after time $t+1$.*

***Remark* 16.1.3** *The weight updates is occurring incrementally. We need two successive predictions to form the error term, so we must already be in state x_{t+1} to obtain the prediction $P_{t,t+1}$. Nonetheless, learning is occurring for prediction $P_{t,t}$ since v_t is being updated. Hence, learning is occurring retroactively for already visited states.*

The author studied the relationship between this heuristic reinforcement signal and the concept of ideal reinforcement signal. Let r^* be the ideal reinforcement signal. The AHC algorithm is based on estimating

$$\begin{aligned} r^*_{t+1} &\approx \sum_{k=1}^{\infty} \gamma^{k-1} \Big[E[r_{t+k} | \, a_t] - E[r_{t+k}] \Big] \\ &= \sum_{k=1}^{\infty} \gamma^{k-1} E[r_{t+k} | \, a_t] - \sum_{k=1}^{\infty} \gamma^{k-1} E[r_{t+k}] \end{aligned}$$

which converges as long as $E[r_{t+k}]$ is bounded. The sequence formed by

$$E[r_{t+k} | \, a_t] - E[r_{t+k}]$$

for successive values of k is called the difference sequence. We are interested in maximising $\sum_{k=1}^{\infty} \gamma^{k-1} E[r_{t+k}]$. These expected values are all implicitly conditional on the structure and state of the environment and the learning system. We denote the state of the environment at time t by q_t and the complete state of the learning system at time t by w_t [2]. Thus, we can rewrite the above equation as

$$r^*_{t+1} \approx \sum_{k=1}^{\infty} \gamma^{k-1} E[r_{t+k} | \, a_t, q_t, w_t] - \sum_{k=1}^{\infty} \gamma^{k-1} E[r_{t+k} | \, q_t, w_t]$$

[2] In the AHC-algorithm, it includes the action-association vector w and the reinforcement-association vector v.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

At time $t+1$, r_{t+1} becomes available for use as an estimate of $E[r_{t+1}|\, a_t, q_t, w_t]$. Using this approximation, and taking the term outside, the above equation becomes

$$r^*_{t+1} \approx r_{t+1} + \sum_{k=2}^{\infty} \gamma^{k-1} E[r_{t+k}|\, a_t, q_t, w_t] - \sum_{k=1}^{\infty} \gamma^{k-1} E[r_{t+k}|\, q_t, w_t]$$

Note, r_{t+k}, for $k \geqslant 2$, does not depend on a_t directly, but only through a_t's effect on q_{t+1}. Thus, the first expected value above can be written as conditional only on q_{t+1}:

$$r^*_{t+1} \approx r_{t+1} + \sum_{k=2}^{\infty} \gamma^{k-1} E[r_{t+k}|\, q_{t+1}, w_t] - \sum_{k=1}^{\infty} \gamma^{k-1} E[r_{t+k}|\, q_t, w_t]$$

Making the change of variable $k \to k+1$ in the first sum, we get

$$r^*_{t+1} \approx r_{t+1} + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} E[r_{t+1+k}|\, q_{t+1}, w_t] - \sum_{k=1}^{\infty} \gamma^{k-1} E[r_{t+k}|\, q_t, w_t]$$

Since the state q_t is imperfectly known, we must rely on the information provided by the stimulus x_t received at time t. So, at best, the heuristic reinforcement signal can approximate

$$r^*_{t+1} \approx r_{t+1} + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} E[r_{t+1+k}|\, x_{t+1}, w_t] - \sum_{k=1}^{\infty} \gamma^{k-1} E[r_{t+k}|\, x_t, w_t]$$

where, in both sums, the complete state of the learning system w is at time t. Further, from Equation (16.1.4), we have the approximation

$$P_{s,t} \approx \sum_{k=1}^{\infty} \gamma^{k-1} E[r_{t+k}|\, x_t, w_s] \tag{16.1.7}$$

***Remark* 16.1.4** *The author conditioned the expectation with respect to w_t, but we condition it with respect to w_s to match Equation (16.1.4).*

To avoid instabilities due to secondary reinforcement creating more secondary reinforcement, special care is taken to separate the time s at which the estimate is made from the time t about which the estimate is made. Here, w may change at each time step, so does $P_{s,t}$ above.

***Assumption* 4** *Changes in w over small time periods, such as those between t and $t+1$, are ignored.*

Thus, replacing the approximated prediction of reinforcement in Equation (16.1.7) in the above equation, the ideal reinforcement signal is approximated by

$$r^*_{t+1} \approx r_{t+1} + \gamma P_{t,t+1} - P_{t,t}$$

which is the heuristic signal in the AHC algorithm in Equation (16.1.6). Note, r^* gives the error in estimating forthcoming reinforcement $P_{s,t}$. If Equation (16.1.7) is exact for both $P_{t,t}$ and $P_{t,t+1}$, then the approximation of r^* is zero. Otherwise, it indicates that the earlier prediction may be too large or too small and should be adjusted accordingly.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

16.1.2.4.2 Link to other models Sutton [1984] extended his model to other tasks, such as the class of time-blind tasks including most board games, and showed link to other models.
In the former, the environment is divided into episodes called episodic task. In time-blind tasks, the goal of learning is defined in terms of performance per episode rather than performance per time step. That is, the goal is to maximise the expected value of the sum of the reinforcement received during the episode:

$$E[\sum_{t=1}^{m} r_t]$$

where m is the length in time steps of the episode. Again, this expectation is implicitly conditional on the task, the learning system, and the state of the learning system. Assuming that a game has reached position q_t, the value of the selected move a_t is defined as the expected value of the sum of the reinforcement received during the rest of the game:

$$E[\sum_{\tau=t+1}^{m} r_\tau | \, a_t, q_t, w_t]$$

where w_t is the state of the learning system and m is the number of moves in the game. Following the same approach as above, the author derived the ideal reinforcement signal. $P_{s,t}$, the estimate at time s of the value of the board position at time t, is

$$P_{s,t} \approx E[\sum_{\tau=t+1}^{m} r_\tau | \, x_t, w_s]$$

and r^* is approximated by

$$r^* \approx \sigma_{t+1} = r_{t+1} + P_{t,t+1} - P_{t,t}$$

which is closely related to the AHC-algorithm. Here, there is no discount rate γ. This is a special case of the AHC-algorithm called the Simplified Samuel's (SS) algorithm due to similarity with Samuel's model (see Samuel [1959]).

Witten [1977] presented an adaptive controller for discrete time Markov environment, such that if the environment moves from state j to state k, then v_j is updated by

$$v_j \leftarrow (1-\beta)v_j + \beta g^{'} \, , \beta \in (0,1)$$

where g is the reward of the environment and $g^{'}$ is its weighted average. The new estimate of future reward is

$$g^{'} = (1-\gamma)g + \gamma v_k$$

where $\gamma \in (0,1)$ is the discount factor. Here, g is equivalent to the primary reinforcement r. Combining the two equations and introducing time indexing, we get

$$v_{j,t+1} = (1-\beta)v_{j,t} + \beta\big((1-\gamma)r_{t+1} + \gamma v_{k,t}\big)$$

We let x_t be an n-vector with zero elements except for the ith one, which is 1, where the environment is in state i at time t. $P_{s,t}$ is given in Equation (16.1.4), then $P_{t,t} = v_{j,t}$ and $P_{t,t+1} = v_{k,t}$. In that setting, the update equation becomes

$$v_{j,t+1} = v_{j,t} + \beta\big((1-\gamma)r_{t+1} + \gamma P_{t,t+1} - P_{t,t}\big)$$

which is a special case of the linear-mapping approach. This equation applies to the v_j corresponding to the state j of the environment at time t. Note, x_t can be used to make this selection, by applying the following equation to all v_i, $1 \leqslant i \leqslant 0$,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$v_{i,t+1} = v_{i,t} + \beta\big((1-\gamma)r_{t+1} + \gamma P_{t,t} - P_{t,t+1}\big)x_{i,t}\ , \forall i$$

Note, in the AHC-algorithm, the trace $\overline{x}_{i,t}$ is used instead of $x_{i,t}$ and the primary reinforcement r is not multiplied by $(1-\gamma)$. Further, σ_t is defined differently. In Witten [1977], it is given by

$$\sigma_{t+1} = (1-\gamma)r_{t+1} + \gamma P_{t,t+1}$$

16.1.2.4.3 A generalisation Sutton [1984] showed that there was similar algorithms to the AHC-algorithm with slightly different ideal reinforcement signal. For example

$$\sigma_{t+1} = r_{t+1} - P_{t,t} + \rho\big(P_{t,t+1} - P_{t,t}\big)$$

where ρ is a positive constant. The author devised a theorem showing that a wide class of other algorithms were equivalent to the AHC-algorithm for particular choice of model parameters.

Theorem 16.1.1 *For any sequence of real numbers $x_{i,t}$, $\overline{x}_{i,t}$, $1 \leqslant i \leqslant n$, and r_{t+1}, $t = 0, 1, 2, ...,$, for σ_{t+1} defined by Equations (16.1.4), (16.1.5), (16.1.6), (the AHC-algorithm), there exists parameters α, β, and γ, such that*

$$\alpha\sigma_{t+1} = \alpha^{'}\sigma^{'}_{t+1}\ , \forall t \geqslant 0$$

where $\sigma^{'}_{t+1}$ is defined by a similar primed set of equations

$$P^{'}_{s,t} = \sum_{j=1}^{n} v^{'}_{j}x_{j,t}$$

$$v^{'}_{i,t+1} = v^{'}_{i,t} + \beta^{'}\sigma^{'}_{t+1}\overline{x}_{i,t}\ , v_{i,0} = 0$$

and

$$\sigma^{'}_{t+1} = ar^{'}_{t+1} + bP^{'}_{t,t+1} - cP^{'}_{t,t}$$

for any constants $\alpha^{'}$, $\beta^{'}$, a, b, c with $\beta^{'} > 0$ and $a, c \neq 0$.

16.1.3 The TD family of learning procedures

The data sequences are generated by absorbing Markov processes, where the next state depends only on the current state. Given an initial state q_1, the absorbing Markov process generates the sequence $q_1, q_2, ..., q_{m+1}$, where $q_{m+1} \in T$. We consider a multi-step problem with experience coming in observation-outcome sequences of the form $x_1, x_2, ..., x_m, z$, where each x_t is a vector of observable input at time t in the sequence, and $z \in \mathbb{R}$ is the outcome of that sequence. Note, z only depends on the terminal state q_{m+1}, that is, z is $\mathcal{F}_T$-measurable. The learning algorithms do not know about the state sequence, but only of the related observation-outcome sequence $x_1, x_2, ..., x_m, z$. The observation vector x_t only depends on the state q_t, and the outcome z (target or payoff) only depends on the terminal state q_{m+1}. We are interested in computing the conditional expectation

$$V(x_t) = E[z|\ x_t] \tag{16.1.8}$$

which we call the value function.

***Remark* 16.1.5** *For simplicity of exposition, the outcome z predicted by the TD-procedure is a scalar. However, Sutton [1989] implemented the case where z is a vector, which we will present in Section (16.2.3.2).*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

16.1.3.1 The $TD(1)$ procedure

A training set, $\mathcal{E}_m$, consists of several sequences $\{e_{m,l}, z\}$, with history (or trajectory)

$$e_{m,l} = \{x_{1,l}, x_{2,l}, ..., x_{m,l}\} \text{ for } l \in [1, L] \tag{16.1.9}$$

For each such sequence, the learner produces a sequence of predictions $P_1, P_2, ..., P_m$, each of which being an estimate of z. For simplicity, it is assumed that P_t is a function of x_t and a vector of weights w, that is, $P_t = P(x_t, w)$.

***Remark* 16.1.6** *While $P_t = P(x_t, w)$ is a function of x_t and w, possibly a perceptron or a two-layers network, it is not a multi-layer network. We discuss combining TD procedures with supervised learning in Section (16.2).*

All learning procedures can be expressed as rules for updating w. We want to find rules for updating the vector w. It is assumed that w does not change during a sequence. For each observation, an increment to w, denoted Δw_t is computed. After a complete sequence has been run, w is modified as follows:

$$w \leftarrow w + \sum_{t=1}^{m} \Delta w_t$$

In supervised learning (see Section (5.1)), each sequence of observations and its outcome is represented by a sequence of observation-outcome pairs $(x_1, z), (x_2, z), ..., (x_m, z)$. In the delta rule (see Section (5.1.4)), the increment due to time t depends on the error between P_t and z, $(z - P_t)$, and the gradient of P_t with respect to w It is given by

$$\Delta w_t = \alpha(z - P_t)\nabla_w P_t \tag{16.1.10}$$

where α is the learning rate. Note, this is Equation (5.1.7) where $(z - P_t)$ represents the scalar error between the prediction P_t and the target value z. In the special case where P is a linear function of x and w, we get $P_t = w^\top x_t = \sum_i w_i x_{t,i}$, where w_i and $x_{t,i}$ are the ith components of w and x_t, respectively. In that setting we have $\nabla_w P_t = x_t$, and we recover the Widraw-Hoff rule

$$\Delta w_t = \alpha(z - w^\top x_t)x_t$$

In the case of backpropagation (see Section (5.1.3)), P_t is computed by a multi-layer network and is a nonlinear function of x_t and w. Here, Equation (16.1.10) still applies, but a complex process is used to compute the gradient $\nabla_w P_t$. Note, all Δw_t depend critically on z, and can only be determined at the end of the sequence, once z is known. Thus, all observations and predictions made during the sequence must be remembered until its end, when all the Δw_t are computed. Thus, Equation (16.1.10) can not be computed incrementally. On the other hand, there is a TD method, called $TD(1)$, producing the same result, but which can be computed incrementally. To do so, we represent the error $z - P_t$ as a change in prediction:

$$z - P_t = \sum_{k=t}^{m} \big(P_{k+1} - P_k\big) \text{ where } P_{m+1} = z$$

We can therefore combine the above equations as

$$\begin{aligned} w \leftarrow w + \sum_{t=1}^{m} \alpha(z - P_t)\nabla_w P_t &= w + \sum_{t=1}^{m} \alpha \sum_{k=t}^{m} \big(P_{k+1} - P_k\big)\nabla_w P_t \\ &= w + \sum_{k=1}^{m} \alpha \sum_{t=1}^{k} \big(P_{k+1} - P_k\big)\nabla_w P_t \\ &= w + \sum_{t=1}^{m} \alpha\big(P_{t+1} - P_t\big) \sum_{k=1}^{t} \nabla_w P_k \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Thus, the rule to use becomes

$$\Delta w_t = \alpha\big(P_{t+1} - P_t\big) \sum_{k=1}^{t} \nabla_w P_k \tag{16.1.11}$$

which can be computed incrementally because each Δw_t only depends the difference between P_{t+1} and P_t, and the sum of past gradients. Hence, we only need to compute one increment to w on each time step.

16.1.3.2 The $TD(\lambda)$ procedure

So far, the predictions in Equation (16.1.11) are altered to an equal extent. The $TD(\lambda)$ procedure permits to make greater alterations to more recent predictions. It is an exponential weighting with recency where the alterations to the predictions of observation vectors occurring k steps in the past are weighted by λ^k for $0 \leqslant \lambda \leqslant 1$. It is given by

$$\Delta w_t = \alpha\big(P_{t+1} - P_t\big) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k \tag{16.1.12}$$

In the case $\lambda = 1$ we recover Equation (16.1.11), hence the name $TD(1)$. One advantage of exponential weighting is that it can be computed incrementally. We denote $e_t = \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$ and write

$$\begin{aligned} e_{t+1} &= \sum_{k=1}^{t+1} \lambda^{t+1-k} \nabla_w P_k \\ &= \nabla_w P_{t+1} + \sum_{k=1}^{t} \lambda^{t+1-k} \nabla_w P_k \\ &= \nabla_w P_{t+1} + \lambda e_t \end{aligned}$$

In the special case where $\lambda = 0$, the weight increment is only determined by its effect on the prediction associated with the most recent observation:

$$\Delta w_t = \alpha\big(P_{t+1} - P_t\big) \nabla_w P_t \tag{16.1.13}$$

This is Equation (16.1.10) with z replaced by P_{t+1}. Both procedures use the same learning mechanism, but with different errors.

16.1.3.3 A random-walk example

As discussed in Section (16.1.1.1), Sutton [1988] showed that TD methods were more efficient than supervised learning methods when the data sequences were generated by a dynamical system. As an example, the author considered a bounded random walk, that is, a random walk (see Appendix (24.6.2)) taking steps to the right or to the left with equal probability, until a boundary is reached. This is illustrated in Figure (16.3), where the walk start in state D and ends if either edge state A or G is reached. A typical walk (path) might be $DCDEFG$.

Figure 16.3: A bounded random walk

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We want to find the conditional probability of a walk reaching state G, given that it is in each of the other states. The outcome is $z = 0$ for a walk ending in state A, and $z = 1$ if it ends in state G. That is,

$$E[z|\, i] \text{ where } z = I_{\{x=G\}}$$

for the walk x and the state i. This is the probability of a right-side termination, that is, the walk reached G before ending in A.
Since the walk corresponds to categorical data we need to perform the one-hot encoding scheme to transform the data into numerical values (see Section (3.1.1.2)). For each nonterminal state i, we associate an observation vector X_i. It is assumed that if the walk is in state i at time t, then $x_t = X_i$. Note, we are no-longer working with the prediction $P_t = P(x_t, w)$ with respect to time, but with the prediction $P_i = P(X_i, w)$ with respect to state i. The vectors $\{X_i\}$ are the unit basis vectors of length 5, which is the size of the walk excluding the bounds. For example, $X_D = (0, 0, 1, 0, 0)^\top$. Thus, the sequence of the above typical walk becomes $X_D, X_C, X_D, X_E, X_F, 1$. The true conditional probabilities for right-side termination are $\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}$ and $\frac{5}{6}$ for states B, C, D, E and F, respectively. That is, the probabilities are linear with respect to the state. This is illustrated in Figure (16.4).

Figure 16.4: Probabilities for a bounded random walk with state $\{A, B, C, D, E, F, G\}$ represented by $\{1, 2, 3, 4, 5, 6, 7\}$.

As a result, this example is particularly suited for a linear prediction model. At time t and state i, for a linear prediction $P_t = w \cdot x_t$, we simply get the value of the ith component of w, denoted w_i. That is, $P_i = w_i$ with gradient $\nabla_w P_i = X_i$.
In that setting, the $TD(0)$ update for w_n after each sequence is given by

$$\begin{aligned} w_{n+1} &= w_n + \sum_{t=1}^{m} \alpha\big(P_{t+1} - P_t\big)\nabla_w P_t \text{ where } P_{m+1} = z \\ &= w_n + \sum_{t=1}^{m} \alpha\big(w_n \cdot X_{q_{t+1}} - w_n \cdot X_{q_t}\big)X_{q_t} \end{aligned}$$

where X_{q_t} is the observation vector corresponding to the state q_t entered at time t within the sequence. Details on an absorbing Markov process are given in Section (16.1.3.4).
The author tested linear supervised learning against TD methods. To get statistically reliable results, 100 training sets, each consisting of 10 sequences, were constructed for use by all learning procedures. The $TD(\lambda)$ procedure in Equation (16.1.12) was used with $\lambda = 0, 0.1, 0.3, 0.5, 0, 7, 0.9, 1$. Two experiments were set:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1. Δw are accumulated over sequences and only used to update the weight vector after the complete presentation of a training set. Each training set are presented repeatedly to each learning procedure until the latter no-longer produce any significant changes in the weight vector. This is the repeat presentations training paradigm.

2. Each training set is presented only once to each procedure rather than repeatedly until convergence. Weight updates are performed after each sequence, rather than after each complete training set. The learning rate parameter α is varied within a range of values. Weight vector is initialised to $\frac{1}{2}$.

The root mean squared (RMS) error was used as a measure of performances. In set [1], averaging over the training sets, performance improved as λ was decreased from 1 to 0. This is illustrated in Figure (16.5).

Figure 16.5: Average error for random walk in case [1]

The Widrow-Hoff procedure, $\lambda = 1$, only minimises error on the training set, but it does not minimises error for future experience. In fact, Sutton proved that $TD(0)$ converges to the optimal estimates for matching future experience, those consistent with the maximum-likelihood estimate of the underlying Markov process.
In set [2], WH produced the worst estimates. This is illustrated in Figure (16.6).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 16.6: Average error for random walk in case [2]

The author also displayed the best error level obtained for each λ value, and the result was not $\lambda = 0$, but it was near 0.3. This is illustrated in Figure (16.7).

Figure 16.7: Average error for random walk in case [3]

This is because $TD(0)$ is slow at propagating prediction levels back along a sequence. There is no problem if the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

sequence is repeatedly presented (case [1]). One solution is to work backwards through the sequences. In the sequence $X_D, X_E, X_F, 1$, first update the prediction for F in light of the 1, then the prediction for E is updated based on F, and so on. Thus, the effect of 1 is back propagated to the start of the sequence with only a single presentation. However, it no-longer has an incremental implementation. But if learning is done offline from an existing database, working backward should produce the best predictions.

16.1.3.4 Convergence

Sutton [1988] proved convergence of the linear $TD(0)$ procedure. He devised an asymptotic convergence theorem for $TD(0)$ when presented with new data sequences (generated by absorbing Markov processes), and presented another theorem stating that $TD(0)$ converges under repeated presentations of optimal estimates.
The data sequences are generated by absorbing Markov processes (such as the random walk process presented in Section (16.1.3.3)), where the next state depends only on the current state. They are defined by a set of terminal states T, a set of nonterminal states N, and a set of transition probabilities p_{ij}, $i \in N, j \in N \cup T$, where each p_{ij} is the probability of a transition from state i to state j, given that the process is in state i. Since it is absorbing, all sequences eventually terminate. Given an initial state q_1, the absorbing Markov process generates the sequence $q_1, q_2, ..., q_{m+1}$, where $q_{m+1} \in T$. The learning algorithms do not know about the state sequence, but only of a related observation-outcome sequence $x_1, x_2, ..., x_m, z$. The observation vector x_t only depends on the state q_t, and the outcome z only depends on the terminal state q_{m+1}.
Similarly to Section (16.1.3.3), it is assumed that there is a specific observation vector X_i corresponding to each nonterminal state i such that, if $q_t = i$, then $x_t = X_i$. For each nonterminal state j, we assume outcomes z are selected from an arbitrary probability distribution with expected value $\overline{z}_j$. We want to match each observation vector X_i to the true expected value of the outcome z given that the state sequence is starting in i. That is, we want the predictions $P(X_i, w)$ to be equal to the conditional expectation $E[z|\, i]$, for all $i \in N$. This is called the ideal predictions. Note, we are no-longer working with the prediction P_t with respect to time, but P_i with respect to state i.
Knowing the Markov process, they are given by

$$E[z|\, i] = \sum_{j \in T} p_{ij}\overline{z}_j + \sum_{j \in N} p_{ij} \sum_{k \in T} p_{jk}\overline{z}_k + \sum_{j \in N} p_{ij} \sum_{k \in N} p_{jk} \sum_{l \in T} p_{kl}\overline{z}_l + \cdots$$

Using matrix notation, we let Q denotes the matrix with entries $[Q]_{ij} = p_{ij}$ for $i, j \in N$, and let h denotes the vector with components $[h]_i = \sum_{j \in T} p_{ij}\overline{z}_j$ for $i \in N$. Then, the above equation rewrites as

$$E[z|\, i] = \Big[\sum_{k=0}^{\infty} Q^k h\Big]_i = \big[(I - Q)^{-1} h\big]_i$$

The Markov properties assure that the probabilities within Q^k converge to 0 as $k \to \infty$. The author introduced the following theorem for the linear $TD(0)$.

Theorem 16.1.2 *For any absorbing Markov chain, for any distribution of starting probabilities μ_i, for any outcome distributions with finite expected values $\overline{z}_j$, and for any linearly independent set of observation vectors $\{X_i|\, i \in N\}$, there exists an $\epsilon > 0$ such that, for all positive $\alpha < \epsilon$ and for any initial weight vector, the predictions of linear $TD(0)$ (with weight updates after each sequence) converge in expected value to the above ideal predictions. That is, if w_n denotes the weight vector after n sequences have been experienced, then*

$$\lim_{n \to \infty} E[X_i \cdot w_n] = E[z|\, i] = \big[(I - Q)^{-1} h\big]_i, \forall i \in N$$

Proof: We refer to Sutton [1988] for the proof of the theorem and just give a few hints. We let m be the number of observation vectors, and write the update for the weight w_n as follows:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} w_{n+1} &= w_n + \sum_{t=1}^{m} \alpha\big(P_{t+1} - P_t\big)\nabla_w P_t \text{ where } P_{m+1} = z \\ &= w_n + \sum_{t=1}^{m-1} \alpha\big(P_{t+1} - P_t\big)\nabla_w P_t + \alpha(z - P_m)\nabla_w P_m \\ &= w_n + \sum_{t=1}^{m-1} \alpha\big(w_n \cdot X_{q_{t+1}} - w_n \cdot X_{q_t}\big)X_{q_t} + \alpha(z - w_n \cdot X_{q_m})X_{q_m} \end{aligned}$$

where X_{q_t} is the observation vector corresponding to the state q_t entered at time t within the sequence. Since each increment corresponds to a particular state transition, we can group them based on the source and destination states of the transitions:

$$w_{n+1} = w_n + \sum_{i \in N}\sum_{j \in N} \eta_{ij}\alpha\big(w_n \cdot X_j - w_n \cdot X_i\big)X_i + \sum_{i \in N}\sum_{j \in T} \eta_{ij}\alpha\big(z - w_n \cdot X_i\big)X_i$$

where η_{ij} denotes the number of times the transition $i \to j$ occurs in the sequence. From independence of the random processes generating the state transitions and outcomes, we take the expected value of each term in the above equation, getting

$$E[w_{n+1}|\, w_n] = w_n + \sum_{i \in N}\sum_{j \in N} d_i p_{ij}\alpha\big(w_n \cdot X_j - w_n \cdot X_i\big)X_i + \sum_{i \in N}\sum_{j \in T} d_i p_{ij}\alpha\big(\overline{z}_j - w_n \cdot X_i\big)X_i$$

where d_i is the expected number of times the Markov chain is in state i in one sequence, so that $d_i p_{ij}$ is the expected value of η_{ij}. For an absorbing Markov chain we have

$$d^\top = \mu^\top\big(I - Q\big)^{-1}$$

where $[d]_i = d_i$ (strictly positive) and $[\mu]_i = \mu_i$, $i \in N$. We let $\overline{w}_n$ be the expected value of w_n. Since the dependence of $E[w_{n+1}|\, w_n]$ on w_n is linear, we can write

$$\overline{w}_{n+1} = \overline{w}_n + \sum_{i \in N}\sum_{j \in N} d_i p_{ij}\alpha\big(\overline{w}_n \cdot X_j - \overline{w}_n \cdot X_i\big)X_i + \sum_{i \in N}\sum_{j \in T} d_i p_{ij}\alpha\big(\overline{z}_j - \overline{w}_n \cdot X_i\big)X_i$$

which is an iterative update formula in $\overline{w}_n$ depending only on initial conditions. Rewriting this update equation in matrix notation and taking the limit $n \to \infty$, after some manipulation, we complete the proof.
As with the Widrow-Hoff procedure, the predictions themselves do not converge. They continue to vary around their expected values according to their most recent experience. In the WH-procedure, the asymptotic variance of the predictions is finite and can be made arbitrarily small by properly choosing the learning rate α. Further, for $\alpha = \frac{1}{n}$, then the variance converges to zero. The author conjectured that the same strong form of convergence also holds for linear $TD(0)$.

16.1.3.5 TD as a gradient descent

Sutton [1988] also explained the fact that TD methods could be viewed as gradient descent in the space of the modifiable parameters (weights). Their goal is to minimise an overall error measure $J(w)$ over the space of weights by repeatedly incrementing the weight vector in an approximation to the direction in which $J(w)$ decreases the most steeply. We let $\hat{\nabla}_w J(w)$ be the approximated gradient, then the rule is

$$\Delta w_t = -\alpha\hat{\nabla}_w J(w)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In a multi-step prediction problem where $P_t = P(x_t, w)$ is supposed to approximates $E[z|\, x_t]$, a natural error measure is the expected value of the square of the difference between these two quantities. That is,

$$J(w) = E_X\big[\big(E[z|\, X] - P(X, w)\big)^2\big] \tag{16.1.14}$$

where X represents the observation vectors. Here, $J(w)$ measures the error for a weight vector averaged over all observation vectors. However, at each time step, we get information about a single observation vector. Thus, the author defined a pre-observation error measure $Q(w, X)$ with the property that

$$E_X[Q(w, X)] = J(w)$$

where

$$Q(w, X) = \big(E[z|\, X] - P(X, w)\big)^2$$

At each time step, the weight increments are then determined by using $\nabla_w Q(w, x_t)$, relying on

$$E_X[\nabla_w Q(w, X)] = \nabla_w J(w)$$

so that the overall effect of the equation for Δw_t can be approximated over many steps using small α as follows:

$$\begin{aligned} \Delta w_t &= -\alpha \nabla_w Q(w, x_t) \\ &= 2\alpha\big(E[z|\, x_t] - P(x_t, w)\big)\nabla_w P(x_t, w) \end{aligned} \tag{16.1.15}$$

Note, the conditional expectation $E[z|\, x_t]$ is not directly known and must be estimated:

- if $E[z|\, x_t]$ is approximated by z (the outcome), then we get the classical supervised learning procedure in Equation (16.1.10).
- if $E[z|\, x_t]$ is approximated by $P(x_{t+1}, w)$ (the following prediction), then we get the extreme TD method, $TD(0)$.

***Remark* 16.1.7** *The key point is that the real goal is for each prediction to match the expected value of the subsequent outcome, not the actual outcome occurring in the training set. This way, we learn about the transition probabilities introduced in Section (16.1.3.4). That is, the actual outcome of a sequence is often not the best estimate of its expected value.*

In simple terms, TD learning is about learning a prediction from a prediction. The update occurs as the system evolves. We learn now, as opposed to waiting a certain time before learning. Rather than matching the training data, TD is concerned with matching future experiences.

16.1.3.6 Intra-sequence weight update

We assumed in Section (16.1.3.1) that the vector of weights w did not change during a sequence, but only after a complete sequence was run. However, since each observation of a sequence generates an increment to the weight vector, Sutton [1988] discussed how to update it immediately after each observation. In order for the update rule that only changes in prediction due to x (and not w) are effective in causing weight alterations, the author proposed the following update:

$$w_{t+1} = w_t + \alpha\big(P(x_{t+1}, w_t) - P(x_t, w_t)\big)\sum_{k=1}^{t} \lambda^{t-k}\nabla_w P(x_k, w_t) \tag{16.1.16}$$

This approach was used by Samuel [1959] in the checker player and Sutton [1984] in the AHC-learning (Adaptive Heuristic Critic).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

16.1.3.7 Predicting cumulative outcomes

Rather that predicting the final outcome z of a sequence, we can also predict cumulative outcomes. We assume that each step of a sequence incur a cost, and want to predict, at time t, the expected total cost of the remaining sequence. Here, the costs become known as the sequence evolves. We use the observation vector received at each step to predict the total cumulative cost after that step. We want P_t to predict the remaining cumulative cost given the t-th observation. We let c_{t+1} be the cost incurred between times t and $t+1$, and $\overline{c}_{ij}$ be the expected value of the cost incurred on transition from state i to state j.

16.1.3.7.1 Finite predictions We assume that the costs are elapsed time. We want P_t to equal the expected value of $z_t = \sum_{k=t}^{m} c_{k+1}$, where m is the number of observation vectors in the sequence. The prediction error (as defined in Equation (5.1.7)) can be expressed in terms of temporal differences as

$$z_t - P_t = \sum_{k=t}^{m} c_{k+1} - P_t = \sum_{k=t}^{m} \sigma_k$$

where

$$\sigma_k = c_{k+1} + P_{k+1} - P_k$$

and $P_{m+1} = 0$ since $\sum_{k=t}^{m}\big(P_{k+1} - P_k\big) = P_{m+1} - P_t$. Following the steps to derive the $TD(\lambda)$ family of procedures in Section (16.1.3.2), the learning rule for the cumulative $TD(\lambda)$ family is given by

$$\Delta w_t = \alpha\sigma_t \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$$

Following Section (16.1.3.4), the ideal prediction for each state $i \in N$ is the expected value of the cumulative sum of the costs $E[z_t|\, x_t = X_i]$. It is computed, as in the $TD(\lambda)$ family, with $\overline{z}$ replaced by $\overline{c}$, so that convergence holds.

16.1.3.7.2 Infinite discounted predictions So far we have considered prediction problems with definite outcomes. In the case of infinite-horizon prediction problems we need to include some discounting. If a process generates costs c_{t+1} at each transition from t to $t+1$, we want P_t to predict the discounted sum

$$z_t = \sum_{k=0}^{\infty} \gamma^k c_{t+k+1}$$

with discounted rate parameter γ, $0 \leqslant \gamma < 1$. We want to find P_t to be equal to z_t and derive a recursive equation describing the relationship between temporally successive predictions. So, if the predictions are accurate, denoted P_t^*, we get

$$\begin{aligned} P_t^* &= \sum_{k=0}^{\infty} \gamma^k c_{t+k+1} \\ &= c_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k c_{t+k+2} \\ &= c_{t+1} + \gamma P_{t+1}^* \end{aligned}$$

Using P_t as an imperfect proxy for the accurate prediction P_t^* (bootstrapping), the TD error is the difference between the two sides of the equation,

$$\sigma_t = \big(c_{t+1} + \gamma P_{t+1}\big) - P_t$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

which is the heuristic reinforcement signal in Equation (16.1.6).

***Remark* 16.1.8** *It is assumed that the predictions are accurate. That is, the prediction are close to the value-function.*

In his Adaptive Heuristic Critic, Sutton [1984] used this error in a learning rule identical to $TD(\lambda)$ given by

$$\Delta w_t = \alpha \sigma_t \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k \tag{16.1.17}$$

where P_t is the linear form $w \cdot x_t$ and $\nabla_w P_t = x_t$. Hence, the AHC-model is best understood as using the linear TD method for predicting discounted cumulative outcomes.

16.1.4 TD revisited and Q-learning

Watkins [1989] revisited the TD-methods of Sutton [1984] [1988] (see Sections (16.1.2) and (16.1.3)) for predicting infinite cumulative return by relating them to the computation of expected returns. The author also discussed a number of different plausible learning methods for Markov decision processes. However, he has only been able to give proof of convergence to the optimal value function and policy in one of them: the one step Q-learning. He then discussed the problem of meta-stable policies and found that the agent must tries all actions in all states to guarantee convergence to an optimal solution. Prior knowledge can be used to induce the agent to search useful actions and visit states of high value. We are briefly going to present these results.
In learning with adaptive transition and reward models, it is assumed that the agent maintain current estimated transition and reward models that are incrementally updated according to experience. He also maintains a current estimated value function, and actions are chosen by look-ahead. The author called primitive learning (PL), the learning where the agent does not have and does not estimate the transition and reward models. He directly develops a policy and evaluation function. PL is described as incremental dynamic programming by a Monte Carlo method. Only the observed state transitions and rewards are used. Thus, the agent must observe the rewards generated by his policy. An episode of the learner's experience consists of the triple $(state, action, reward)$, that is, $\{s_1, a_1, r_1, ..., s_n, a_n, r_n\}$. The value function $V_\pi(s)$ is the expected value of the actual returns after starting in state s with policy π. We let P be the approximation of the current value function. As time t evolves, the agent seeks to make P approximate the value function more closely. Thus, it will change with time. At time t, P is denoted by P_t. This is to relate to the prediction of reinforcement forthcoming after time t, made by the learning system at time s, denoted by $P_{s,t}$ and defined in Equation (16.1.4).

16.1.4.1 Estimating expected returns

All methods of PL rely on estimating values and action-values from experience. At all times, the agent has a policy and estimates the values and action-values with that policy. He must therefore estimate the expected returns. The actual return (discounted cumulative returns) from time t are

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n r_{t+n} + \cdots$$

where r_t is the reward received at time t. This is the sum of discounted rewards.

***Remark* 16.1.9** *The first reinforcement or cost in the actual return by Sutton [1984] [1988] is r_{t+1} and not r_t.*

The optimal policy, if followed in perpetuity, will optimise the expected actual return from every state. Thus, we want to optimise the actual returns from every state. Since $\gamma < 1$, γ^n approaches zero as n gets large. Thus, there is a number of time-steps n after which the remaining part of the returns becomes negligible. The agent can therefore compute an accurate value of the actual return from time t at time $t + n - 1$. The n-step truncated return is

$$R_t^{[n]} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-1} r_{t+n-1}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Since returns are bounded, the maximal difference between the truncated return and actual return is bounded, and tends to zero as n is increased. We can therefore estimate the value-function by following a policy and averaging its n-step truncated returns for n large enough. However, the value of truncated return is only revealed to the agent after a time delay of n steps and it forces the agent to use policy π for at least n steps after t.
As an alternative, Sutton [1984] proposed another class of estimators, called the temporal difference methods, that rely on the agent to maintain a current approximation to the value-function. The truncated rewards ignore the terms $\gamma^n r_{t+n} + \gamma^{n+1} r_{t+n+1} + \cdots$ that would be received after n steps. On the other hand, the sum

$$r_{t+n} + \gamma r_{t+n+1} + \cdots$$

can be approximated by the agent as $P_{t+n}(s_{t+n})$. It can be used as a correction term for the n-step truncated return. That is, the corrected n-step truncated return for time t is

$$R_t^{(n)} \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n P_{t+n}(s_{t+n}) \tag{16.1.18}$$

If P was equal to V_π, then the corrected truncated returns would be unbiased estimators of V_π. This is because the expected value of the corrected truncated return tends to be closer to V_π than P is. Let K be the maximum absolute error of P, given by

$$K = \max_s |P(s) - V_\pi(s)|$$

Then the error-reduction property of corrected truncated returns is

$$\max_s |E[R^{(n)}(s)] - V_\pi(s)| \leqslant \gamma^n K$$

Rather than using $R^{(n)}$, we can use weighted averages of $R^{(n)}$ for different values of n. We let $R^{(w)}$ be the weighted sum of corrected truncated returns

$$R^{(w)} = \sum_i w_i R^{(i)}$$

where the weights w_i sum to one. Then the error-reduction property is

$$\max_s |E[R^{(w)}] - V_\pi(s)| \leqslant \sum_i \gamma^i |w_i| K$$

Sutton [1984] [1988] proposed to have $R^{(n)}$ be proportional to λ^n for some λ in $[0, 1]$. This weighted sum is denoted by R^λ. In that setting, the $TD(\lambda)$ return from time t is

$$\begin{aligned} R_t^\lambda &= (1-\lambda)\big(R_t^{(1)} + \lambda R_t^{(2)} + \lambda^2 R_t^{(3)} + \cdots\big) \\ &= r_t + \gamma(1-\lambda)P_t(s_{t+1}) + \\ &\quad \gamma\lambda\big[r_{t+1} + \gamma(1-\lambda)P_{t+1}(s_{t+2}) + \\ &\quad \gamma\lambda\big[r_{t+2} + \gamma(1-\lambda)P_{t+2}(s_{t+3}) + \\ &\quad \gamma\lambda\big[r_{t+3} + \gamma\big[\cdots \end{aligned}$$

***Remark* 16.1.10** *P is time-indexed. From Equation (16.1.18), we should get terms $P_{t+k}(s_{t+k})$, $k = 1, 2, ..$, and not terms $P_{t+k-1}(s_{t+k})$. However, Assumption (4) applies. This is $P_{t-1,t}$ in Equation (16.1.7).*

Note, P may change slightly at each time step, but these changes will be small if the learning factor is small. The above expression may be rewritten recursively as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$R_t^\lambda = r_t + \gamma(1-\lambda)P_t(s_{t+1}) + \gamma\lambda R_{t+1}^\lambda \tag{16.1.19}$$

Setting $\lambda = 0$, we get the $TD(0)$ return

$$R_t^0 = r_t + \gamma P_t(s_{t+1})$$

and setting $\lambda = 1$, we get the $TD(1)$ return

$$R_t^1 = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

which is the actual return.
The choice of λ depends on a trade-off between bias and variance. If the values of P are close to those of V, then the variance will be lowest for $\lambda = 0$ and highest for $\lambda = 1$. Since low variance is obtained by truncating the sequence of discounted returns and adding the correction term P, if the latter is wrong, the estimates will be biased. Thus, the optimal choice of λ depends on the difference between P and V_π. One solution is to start with λ close to 1 and decrease it to 0 as P gets closer to V_π. Sutton [1984] defined the prediction difference at time t as

$$e_t = r_t + \gamma P_t(s_{t+1}) - P_t(s_t)$$

which corresponds to the heuristic reinforcement signal in Equation (16.1.6).

Claim 2 *The main idea is that at time t, the agent might predict the return from time t to be $P_t(s_t)$. At time $t+1$, the agent has more information for predicting the return from time t, since he can observe r_t and s_{t+1}. He may therefore make a new prediction of the return from time t (one step backward) as*

$$r_t + \gamma P_t(s_{t+1})$$

where P_t is seen from time t. So, we have two predictions of the return from time t. This is based on Assumption (4) and $P_t(s_{t+1})$ correspond to $P_{t,t+1}$ in Equation (16.1.4). See Section (16.1.2).

The prediction difference is the difference between these two predictions made at time t. If P is equal to V_π for all states, then the expected value of all prediction differences is zero. The author wrote the difference between the $TD(\lambda)$ return and the estimated value as

$$\begin{aligned}
R_t^\lambda - P_t(s_t) &= r_t + \gamma(1-\lambda)P_t(s_{t+1}) + \gamma\lambda R_{t+1}^\lambda - P_t(s_t) \\
&= e_t + \gamma\lambda\big[R_{t+1}^\lambda - P_{t+1}(s_{t+1}) + P_{t+1}(s_{t+1}) - P_t(s_{t+1})\big] \\
&= e_t + \gamma\lambda e_t + +\gamma\lambda\big(P_{t+1}(s_{t+1}) - P_t(s_{t+1})\big) + (\gamma\lambda)^2\big[R_{t+2}^\lambda - P_{t+2}(s_{t+2}) + P_{t+2}(s_{t+2}) - P_{t+1}(s_{t+2})\big] \\
&= e_t + \gamma\lambda e_{t+1} + \gamma^2\lambda^2 e_{t+2} + \cdots + \sum_{n=1}(\lambda\gamma)^n\big(P_{t+n}(s_{t+n}) - P_{t+n-1}(s_{t+n})\big)
\end{aligned}$$

If the learning factor is small, then the second summation on the RHS will be small. Thus, the agent can compute the prediction difference for time t at time $t+1$, with a delay of only one time step.
The natural way of using R^λ for improving the estimated value function P is to consider the following update rule

$$\begin{aligned}
P_{t+1}(s_t) &= (1-\alpha)P_t(s_t) + \alpha R_t^\lambda \\
&= P_t(s_t) + \alpha\big(R_t^\lambda - P_t(s_t)\big) \\
&= P_t(s_t) + \alpha\big(e_t + \gamma\lambda e_{t+1} + \gamma^2\lambda^2 e_{t+2} + \cdots\big)
\end{aligned}$$

At each time step we add the appropriate fractions of the current prediction difference to previously visited states. It was implemented by Barto et al. [1983], who maintained an eligibility trace, a trace of how eligible the estimated

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

value of a state is to be modified. Watkins [1989] called it the activity trace. The activity is high when a state is visited, and it declines gradually afterwards. We denote $C(s,t)$ the activity of a state s at time t. The levels of activity of all states are updated at each time step as follows:

$$C(s,t) = \begin{cases} 0 \text{ if s has never been visited} \\ \gamma\lambda C(s,t-1) \text{ if } s_t \neq s \\ 1 + \gamma\lambda C(s,t-1) \text{ if } s_t = s \end{cases}$$

Here, 1 is added to the activity of the current state, and the activities of all states decay exponentially with a time constant of $\gamma\lambda$. Sutton suggested the following update rule

$$P_{t+1}(s) = P_t(s) + \alpha C(s,t+1)e_t$$

It is suitable for connectionist implementations of incremental updating of value functions, where all states are updated at each time step. When using sequential computers, it is more efficient to keep track of the last n states visited for sufficiently large n. The size of n will depend on the value of λ. For $\lambda = 0$, we get $n = 1$.

***Remark* 16.1.11** *Sutton [1984] presented the update rule for the action-association vector w in Equation (16.1.3), which relies on the trace of some eligibility terms. In the AHC-algorithm, the prediction of reinforcement forthcoming after time t, P_t, is given in Equation (16.1.4), which is linear in the stimuli and the reinforcement-association vector. The update of the prediction of reinforcement P is done via the update of the reinforcement-association vector v in Equation (16.1.5), where the trace is applied to the stimuli. Here, the stimuli corresponds to the gradient of P with respect to the reinforcement-association vector v (see Equation (16.1.12)).*

16.1.4.2 Estimating and learning the action-value function

16.1.4.2.1 Estimating the Q-function In the previous section we have estimated the expected returns in a Markov process with reward. However, in a decision process, where different actions are possible at each state, we can only estimate returns relative to a policy. In the method described above, the agent estimates returns relative to an internal policy, its estimation policy. The latter can be stochastic, or the agent may keep changing policy. In that setting, Watkins [1989] showed that we can still estimate the value function by using different value of λ in the $TD(\lambda)$ return at different time steps, with the value of λ depending on whether the current action is a policy action or not. Similarly to $TD(\lambda)$ return defined in Equation (16.1.19), the author defined

$$\begin{aligned} R_t^\Lambda &= r_t + \gamma(1-\lambda_{t+1})P_t(s_{t+1}) + \gamma\lambda_{t+1}R_{t+1}^\Lambda \\ &= (1-\lambda_{t+1})R_t^{(1)} + \lambda_{t+1}(1-\lambda_{t+2})R_t^{(2)} + \lambda_{t+1}\lambda_{t+2}(1-\lambda_{t+3})R_t^{(3)} + \cdots \end{aligned} \tag{16.1.20}$$

where Λ is a sequence of values $\lambda_1, \lambda_2, ...$, and where $0 \leqslant \lambda_t \leqslant 1$ for all t. Note, λ_t may depend on the state and action at time t. Under a suitable choice of λ_t, the agent may estimates the returns under one policy, while behaving differently. The agent can classify each step of his experience either as a policy or as experiment. His estimated returns should not be contaminated by the effects of experimental actions. In that case, there are several ways to defining Λ. For instance, one of it is as follows:

$$\lambda_t = \begin{cases} \lambda^* \text{ if } a_t \text{ is a policy action} \\ 0 \text{ if } a_t \text{ is an experimental action} \end{cases}$$

where $\lambda^* \in [0,1]$. Here, if step t is experimental, λ_t must be zero. Thus, in that setting, R_t^Λ is an estimate of the action-value function $Q_t(s_t,a_t)$. It is demonstrated by considering the recursive equation of R_t^Λ in Equation (16.1.20). In the case of an experimental action at step $t+1$, we get $\lambda_{t+1} = 0$, so that $\gamma\lambda_{t+1}R_{t+1}^\Lambda$ is zero. As a result, R_t^Λ is an estimate of $Q_\pi(s_t,a_t)$, since after step t, it is constructed from $P_{t+1}(s_{t+1})$ and from any further policy steps from $t+1$ onwards. If some step $t+n$ is an experiment, we get $\lambda_{t+n} = 0$, and no further time steps contribute to the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

estimate. Thus, R_t^Λ is an estimate of the action-value of (s_t, a_t). If a_t is experimental we can no-longer use R_t^Λ to estimate $V_\pi(s_t)$. The author defined P_t^Λ as

$$P_t^\Lambda = (1-\lambda_t)P_t(s_t) + \lambda_t R_t^\Lambda$$

so that $V_\pi(s_t)$ is estimated only if the action at time t is a policy action. This is a special case of variance reduction in Monte Carlo estimation of Markov processes.

16.1.4.2.2 Learning the Q-function As discussed in Section (15.3.3), learning optimal value function is a process of simultaneously improving a value function and policy. One solution is to store action-values $Q(s,a)$ for each state s and action a. We let Q_t be the values of Q at time t. From its values of Q, the agent can estimate the value of a state s as

$$P_t^Q(s) = \max_{a\in\mathcal{A}} Q_t(s,a)$$

where P^Q is calculated from Q. From Definition (15.3.7) and Proposition (5), Q defines the current policy π as

$$\pi_t = a \text{ such that } Q_t(s,a) = P_t^Q(s)$$

The policy is to choose the actions with maximal estimated action-value. The agent can improve Q via experience by using the one-step Q-learning, where Q is updated as follows:

$$\begin{aligned} Q_{t+1}(s_t,a_t) &= (1-\alpha)Q_t(s_t,a_t) + \alpha\big(r_t + \gamma P_t^Q(s_{t+1})\big) \\ &= (1-\alpha)Q_t(s_t,a_t) + \alpha R_t^{(1)} \end{aligned} \tag{16.1.21}$$

where α is a learning factor and $R_t^{(1)}$ is calculated from Q. The values of Q for all other combinations of s and a are left unchanged. Here, the agent must try each action in each state many times. It can be viewed as incremental Monte Carlo value iteration: Q_{t+1} is estimated from P_t^Q, which is itself obtained by maximising Q_t at each state. The author proved convergence for finite Markov decision processes.
In fact, there is a family of methods of Q-learning that uses different estimators of expected returns, where the values of Q are updated by using estimates R_t^Λ of the action values. Given λ_t defined as above, returns are estimated only over sequences of policy actions. A softer version let λ_t depend on the difference between the estimated action value of the action performed and the estimated value of the state. For example, it is given by

$$\lambda_t = e^{-\eta\left(P_t^Q(s_t) - Q_t(s_t,a_t)\right)}\lambda^*$$

where η is a non-negative real number. If η is zero, then $\lambda_t = \lambda^*$ for all t. Since R_t^Λ only becomes available at some later time, Q can only be updated at a later time denoted $t+T$ as follows:

$$Q_{t+T+1}(s_t,a_t) = (1-\alpha)Q_{t+T}(s_t,a_t) + \alpha R_t^\Lambda$$

Alternatively, it may be performed incrementally by using the method of prediction differences described above. However, if Q is incrementally updated at each time step, it is also used in computing the estimates of returns R^Λ on which the updates are based. That is, changes in Q may affect R^Λ, which then affect the changes in Q, and so on. Fortunately, these effects are proportional to α^2, and are negligible for small α. Note, the author tried other methods as well, but he did not prove that these methods were converging to an optimal solution.
So far, action-weights have been represented, but action-values have not been represented explicitly.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

16.1.4.2.3 Convergence of one-step Q-learning We provide some hints at the proof of convergence of the one-step Q-learning given by Watkins [1989]. We assume that we have an infinite sequence of observations $e = (s, a, r, s^{'})$ used to update Q, which are independent of state transitions and rewards in a Markov process. The nth observation is denoted $e_n = (s_n, a_n, r_n, s_n^{'})$. The author constructed a notional Markov decision process from the data. The initial values of Q are denoted $Q_0(s, a)$ for each state s and action a. The estimated value of a state s at the nth step is

$$P_n(s) = \max_{a \in \mathcal{A}} Q_n(s, a)$$

Given the nth observation e_n, we compute Q_n from Q_{n-1} by

$$Q_n(s, a) = \begin{cases} (1 - \alpha_n)Q_{n-1}(s, a) + \alpha_n\big[r_n + \gamma P_{n-1}^{Q}(s_n^{'})\big] \text{ if } s = s_n \text{ and } a = a_n \\ Q_{n-1}(s, a) \text{ otherwise} \end{cases}$$

where the learning factor α_n may depend on s_n and a_n.
The action replay process (ARP) is a purely notional Markov decision process used for the proof consisting of layers of states, numbered $0, 1, 2, ..., n, ...$ In the kth layer, there is a state $< s, k >$ corresponding to each state s in the RP. It is assumed that an action in the ARP is a replay of an observation. That is, performing an action a in state s is simulated by recalling an observation e and then using the observed r and $s^{'}$ as the simulated reward and new state, respectively. For instance, when at state $< s, k >$ in the APR, if we want to perform action a, we must find the first k observations to replay. An observation is eligible for replay if it was observed observation k and if it is of the form $(s, a, \cdot, \cdot)$ where (s, a) correspond to the state $< s, k >$ we are at, and the action we want to perform. Given the lth observation $(s, a, r_l, s_l^{'})$, if it is eligible for replay, the reward is r_l and the next state reached is $< s_l^{'}, l - 1 >$. We can then repeat the process and choose to perform action b and look for an observation to replay. However, only the first l observations are eligible. For each action taken in the APR, the list of eligible observations shorten until there is no more eligible observation. In that case we get the final payoff $Q_0(z, c)$ where z and z are the state we are at the the action we want to perform when running out of actions to replay, respectively. We therefore have a finite number of actions to perform in the APR.
The observations are randomly selected for replay. Given the current state $< s, k >$, we want to perform action a with the eligible observations

$$n_1 < n_2 < \cdots < n_i \leqslant k$$

The probability of replaying observation n_i is α_i, the probability of replaying observation n_{i-} is $(1 - \alpha_i)\alpha_{i-1}$, and so on. The probability of not replaying any of the eligible action is

$$(1 - \alpha_i)(1 - \alpha_{i-1}) \cdots (1 - \alpha_1)$$

If no eligible action is selected for replay, the APR terminates with final payoff $Q_0(s, a)$. The decision to perform action a in state $< s, k >$ is based on a recursive algorithm.
The return of a sequence of replays of observations $k_1, k_2, ..., k_n$ (such that $k_1 > k_2 > \cdots > k_n$) is

$$r_{k_1} + \gamma r_{k_2} + \cdots + \gamma_n r_{k_n} + \gamma_{n+1} Q_0(s_{k_n}^{'}, a)$$

where a is the action chosen in the last state $< s_{k_n}^{'}, k_n - 1 >$ reached before the APR terminates. To show that Q_n is the optimal action-values for the APR at step n, we let $Q_{APR}^{*}(< s, n >, a)$ be the optimal action-value for action a at state $< s, n >$ of the APR, and P_{APR}^{*} is the optimal value function of the APR. The author proved by induction the following action-replay theorem:

Theorem 16.1.3 *For all (s, a), $Q_n(s, a)$ is the optimal action-value at step n of the APR. That is,*

$$Q_n(s, a) = Q_{APR}^{*}(< s, n >, a)$$

for all (s, a), and for all $n \geqslant 0$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Then, the author discussed the convergence of Q^*_{APR} to Q^* as $n \to \infty$. Sufficient conditions are that for each state-action pair (s, a)

- There is an infinite number of observations of the form $(s, a, r_n, s_n^{'})$.
- The learning factors α_n for observations of the form $(s, a, r_n, s_n^{'})$ are positive, decrease monotonically with increasing n, and tend to zero as $n \to \infty$.
- The sum of the learning factor α_n for observations of the form $(s, a, r_n, s_n^{'})$ is infinite.

16.1.5 Finite time performance

16.1.5.1 Overview

We saw in Section (16.1.3) that Sutton [1988] proved convergence of $TD(0)$. Then, Dayan [1992] and Bertsekas et al. [1996] proved asymptotic convergence of $TD(\lambda)$. They showed that the TD algorithms could be viewed as a special case of linear stochastic approximation (see Section (15.3.4.3)). As a result, finite-time bounds for linear stochastic approximation can be converted to finite-time bounds on TD algorithms. This led some authors to study finite time performance. For instance, Dalal et al. [2017], Lakshminarayanan et al. [2018] studied the finite-time performance of the TD algorithm, by assuming the data to be independent and identically distributed (i.i.d.). Assuming that the standard algorithm includes a projection step, Bhandari et al. [2018] found bounds on the finite-time mean-square error in the general Markovian case. Srikant et al. [2019] considered the dynamics of a linear stochastic approximation model driven by Markovian noise and derived finite-time bounds on the moments of the error. That is, the deviation of the model outputs from the equilibrium point of an associated ordinary differential equation (ODE). Using these results, Gupta et al. [2019] obtained finite-time bounds for the performance of TD learning algorithms with linear function approximation and a constant step size, without requiring a projection step or an i.i.d. noise assumption.

16.1.5.2 Adaptive learning rate

The convergence of TD learning can vary significantly based on the learning rate used. Some existing learning rate rules are: diminishing learning rates, constant learning rate, and learning rate schedules. Assuming a linear function approximation, Gupta et al. [2019] chose the learning rate adaptively by observing the evolution of the function parameters as the algorithm evolves. They followed Srikant et al. [2019], who derived finite-time bounds on the moments of the error. They considered the linear stochastic recursion with a constant step size α given by

$$w_{k+1} = w_k + \alpha\big(A(x_k)w_k + b(x_k)\big)$$

where w_k is a random vector, $A(\cdot)$ is a random matrix and $b(\cdot)$ is a random vector. $\{x_k\}$ is the underlying Markov chain. They assumed the following two limits:

$$\lim_{k\to\infty} E[A(x_k)] = \overline{A} \text{ and } \lim_{k\to\infty} E[b(x_k)] = 0$$

The corresponding ODE for the above stochastic recursion is

$$\dot{w} = \overline{A}w$$

The matrix is assumed to be a Hurwitz matrix (all eigenvalues have strictly negative real parts), so that the equilibrium point of the ODE is 0. This recursion is called linear stochastic approximation. The $TD(\lambda)$ can be rewritten in this form.

Following the notation in Section (16.1.3.3), the prediction function is $P_t = P(x_t, w)$ with respect to time. We assume that P is a linear function of x and w, that is, $P_t = w^\top x_t = \sum_i w_i x_{t,i}$, where w_i and $x_{t,i}$ are the ith components of w and x_t, respectively. The size of the state space is denoted by N and the dimension of w by d, with $d \leqslant N$. For each nonterminal state i, we associate an observation vector X_i. It is assumed that if the chain is in state i at time t,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

then $x_t = X_i$. The authors denoted it $\phi(i)$, which is a feature vector associated with state i. The update rule in the TD algorithm is

$$w_{k+1} = w_k - \alpha\big(w_k \cdot \phi(x_k) - c(x_k) - \gamma w_k \cdot \phi(x_{k+1})\big)\phi(x_k)$$

where w_k is the estimate of w at time k, and $\alpha \in (0,1)$ is the learning rate. It can be rewritten in the more convenient form

$$w_{k+1} - w^* = w_k - w^* + \alpha\big(A(x_k)(w_k - w^*) + b(x_k)\big)$$

where w^* is the centering offset from 0. The equilibrium point of the ODE is

$$\frac{dw}{dt} = E[A(x_\infty)]w$$

where x_∞ is a random variable with a distribution equal to the stationary distribution of the Markov chain x. The authors established the following result:

Theorem 16.1.4

$$E[\|w_k - w^*\|] \leqslant K_1(1 - K_2\alpha)^k + K_3\alpha$$

for all sufficiently small α and large k, where K_1, K_2, and K_3 are problem dependent constants.

We are interested in making the learning rate α a function of time steps to minimise the mean-squared error for a given number of samples. Assuming time-varying learning rates, we could optimise them to minimise the error for the given number of samples. But it is computationally intractable. In order to find a simple adaptive learning rate rule, Gupta et al. [2019] considered a time T such that the error due to the transient and steady-state parts in the above theorem are equal:

$$K_1(1 - K_2\alpha)^T = K_3\alpha \tag{16.1.22}$$

Once reached T, the learning rate α will no-loner improve the TD algorithm and the mean-squared error is bounded by twice the steady-state error $K_3\alpha$. Hence, at time T, we can reset the learning rate as $\alpha \leftarrow \frac{\alpha}{\xi}$, where $\xi > 1$ is a constant used to reduce the learning rate. However, this approach relies on estimating the time T. One solution is to plot $\|w_k - w^*\|$ as a function of k to see when the curve is close enough to a steady state. Another solution is to observe the measure $\|w_k - w_0\|$, where w_0 is an initial guess for w^*. See Figure (16.8).

Figure 16.8: Learning rate curve for the measure $\|w_k - w_0\|$.

At $k = 0$, the measure is zero, and it will increase to $\|w^* - w_0\|$. We then need to devise if that measure has reached a steady-state. Using the above theorem and the steady-state time in Equation (16.1.22), the authors derived the inequality

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$c - \sqrt{2K_3\alpha} \leqslant E[\|w_k - w_0\|] \leqslant c + \sqrt{2K_3\alpha}$$

For $k \geqslant T$, we consider the N points $\{x_i = i,\ y_i = \|w_{k+i} - w_0\|\}_{i=1}^N$, all obtained after the steady state is reached. Drawing the best-fit line through these points, the slope should be small. Using the above inequality and the formulas for the slope in linear regression, the authors got

$$E[|\psi_N|] = O(\frac{\sqrt{\alpha}}{N}) \,, Var(\psi_N) = O(\frac{1}{N^2})$$

Hence, if $N \geqslant \frac{k}{\sqrt{\alpha}}$, then the slope of the best-fit line connecting $\{x_i, y_i\}$ will be $O(\frac{\sqrt{\alpha}}{N})$ with high probability. Further, when we are in the transient state, the difference between $\|w_{k+m} - w_0\|$ and $\|w_k - w_0\|$ will be $O(m\alpha)$. Thus, the slope of the best-fit line through N consecutive points in the transient state are $O(\alpha)$. Since we chose $N \geqslant \frac{k}{\sqrt{\alpha}}$, the slope of the best-fit line in steady state, $O(\frac{\sqrt{\alpha}}{N})$, will be lower than the slope of the best-fit line in the transient phase, $O(\alpha)$ (for sufficiently large k). Using these results, the authors devised a diagnostic test to determine if the algorithm has entered a steady state. See Algorithm (33)). If the test is true, then the earning rate is updated.

Algorithm 33 Adaptive Learning Rate Rule

Require: the hyperparameters α_0, w_0, σ, ξ, N
Require: initialise $\alpha = \alpha_0$, $\psi_N = 2\sigma\sqrt{\alpha}$
for $i = 1, 2, ...$ **do**
  Do TD learning update
  Compute $\psi_N = \left(\{k, \|w_{i-k} - w_0\|\}_{k=0}^{N-1}\right)$
  if $|\psi_N| < \frac{\sigma\sqrt{\alpha}}{N}$ **then**
    $\alpha = \frac{\alpha}{\xi}$
    $w_0 = w_i$
  end if
end for

16.2 Combining supervised learning and TD procedures

16.2.1 Overview

Tesauro [1992] studied the $TD(\lambda)$ algorithm in the case of complex, stochastic, nonlinear real-world problems, and noted some practical issues. For instance, since the $TD(\lambda)$ algorithm applies to look-up table representation, it is intractable for real-world problems, where every possible state in the state space has to be visited infinitely many times to guarantee convergence (see Section (16.1.4.2)). The solution proposed was to use multilayer neural networks to learn more complex nonlinear functions.
Both backpropagation (see Section (5.1.3)) and TD-learning methods (see Section (16.1.3)) are designed to accurately assign credit. That is,

- Backpropagation: it decides which part of a network to change in order to improve convergence to the output by reducing the overall error of a loss function. It addresses a structural credit-assignment problem.
- TD methods: they decide how each output of a temporal sequence of outputs should be changed. This is done by assigning blame for error over the sequence of predictions made by the learning agent. They address a temporal credit-assignment problem.

Both methods are perfectly compatible and can easily be combined. Recall, neural networks generalise learning across similar states. Thus, neural networks can be used as an approximation scheme to store the value function.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Hence, by combining TD and backpropagation, we can learn to maximise reward over multiple time steps and capture the structural similarities in input patterns.

While Sutton [1988] discussed the linear case for TD methods, they are applicable for predictions $P(x_t, w)$ formed by nonlinear functions. The key requirement being that the gradient $\nabla_w P_t$ is computable, which is the case with backpropagation. Hampson [1983] developed a learning algorithm similar to the AHC algorithm with a method for learning in hidden units and for learning in an action network. Anderson [1986] [1987] implemented such a combination of backpropagation and TD method for solving a nonlinear broomstick-balancing task and the Towers of Hanoi problem. Combining the two methods, Tesauro [1992] devised a program that learned to play backgammon better than supervised learning. The networks have a feedforward fully connected architecture with either no hidden units, or a single hidden layer with between 10 and 40 hidden units. They were trained on the entire game, starting from the opening position all the way to the end. The TD nets could learn on their own to play at a better level than Gammontool and could even surpass the EP nets trained on a massive human expert data base. The author concluded that, empirically the $TD(\lambda)$ algorithm always converged to at least a local minimum, and the quality of solution improved with increasing network size. Further, the scaling of training time with the length of input sequences, and with the size and complexity of the task, was not a serious problem. At last, with no built-in knowledge, the TD nets could outperform identical networks trained on a massive data base of expert examples.

McClelland [2015] described a simple implementation of the temporal difference backpropagation (TDBP), which combines TD procedures with neural networks. We are going to present the implementation of the TDBP.

The predictive representations (PRs) are methods for learning generative models of dynamical systems (see Appendix (23.4)). The main idea being that knowledge about the world can be represented in terms of predictions of relationships between the observable quantities. For instance, Rivest et al. [1990], Jaeger [1998], Littman et al. [2002] proposed different methods for learning a predictive representation of state. Tanner et al. [2005a] introduced the TD-networks as a formalism for expressing and learning knowledge in a predictive form. Tanner et al. [2005b] extended the model to learn over longer time scales by using a $TD(\lambda)$-like algorithm. Maei et al. [2009] proposed the gradient temporal-difference (GTD) learning, which minimises an objective function corresponding to the error in the Bellman equation. Silver [2012] combined the GTD with backpropagation through time (BPTT), or, real-time recurrent learning (see Williams et al. [1989]), obtaining a generalisation of the Bellman equation corresponding to the semantics of TD networks.

16.2.2 Multilayer connectionist system

Following Barto et al. [1983] and Sutton [1984], Anderson [1986] presented a connectionist learning system to learn strategies from experience. Rather than considering networks with single units, the approach was extended to multilayer networks. The system is made of a two-layer connectionist network, called the evaluation network, that learns an evaluation function with the AHC-algorithm, and simultaneously, a reinforcement-learning algorithm is used in a second network, called the action network, to learn the mapping from states to actions. This is illustrated in Figure (16.9).
Input from the environment is represented by $x_0, x_1, ..., x_n$, and is provided to all hidden units and output units. There is an interconnection weight at every intersection: hidden units receive $n+1$ inputs and have $n+1$ weights each. The output units receive $n+1+m_h$ inputs and have $n+1+m_h$ weights. The weights of the evaluation network are denoted v_{ij} for the ith input to Unit j, and are denoted w_{ij} in the action network. When referenced to the time step t, they are denoted by $x_{k,t}$, $v_{i,j,t}$ and $w_{i,j,t}$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 16.9: Two-layer networks: evaluation and action networks.

16.2.2.0.1 The evaluation network The evaluation network consists of the AHC-algorithm for the output unit and the error backpropagation algorithm by Rumelhart et al. [1985] [1986b] for the hidden units. The former results in a prediction of future reinforcement for a given state, which guide the search in the action network.
The output p_j of Hidden Unit j is calculated by using the values of its weights v at time t_v and input x at time t_x:

$$p_j(t_x, t_v) = f\big(\sum_{i=0}^{n} x_{i,t_x} v_{i,j,t_v}\big) \text{ for } j \in H$$

where f is the logistic function. The input vector y for the output unit is composed of the input from the environment and the output of the hidden units:

$$\begin{aligned} y_i(t_x, \cdot) &= x_{i,t_x} \text{ for } i = 0, ..., n \\ y_i(t_x, t_v) &= p_{i-n}(t_x, t_v) \text{ for } i = n+1, ..., n+m_h \end{aligned}$$

The index m of the single output unit is dropped from p_m for clarity. The output of the evaluation network is p, given by

$$p(t_x, t_v) = \sum_{i=0}^{n+m_h} y_i(t_x, t_v) v_{i,m,t_v}$$

In order to update the output unit, we compute the heuristic reinforcement σ_t given by

$$\sigma_t = r_t + \gamma p(t, t-1) - p(t-1, t-1)$$

where γ is the discount rate. It results in the prediction of future discounted reinforcement for the current state. States for which p is relatively large are favourable, while the low ones are avoided. Sutton [1984] specialised the AHC algorithm by defining σ_t for several classes of tasks involving distinct trials. Following the same approach, the author defined

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\sigma_t = \begin{cases} 0 \text{ if state at time t is a start state} \\ r_t - p(t-1,t-1) \text{ if state at time t is a goal or failure state} \\ r_t + \gamma p(t,t-1) - p(t-1,t-1) \text{ otherwise} \end{cases}$$

The weights of the output unit, Unit m, are updated as follows:

$$v_{i,m,t} = v_{i,m,t-1} + \beta \sigma_t y_i(t-1,t-1)\,,\, i = 0,...,n+m_h$$

for $\beta > 0$. A positive σ_t is a positive change in state evaluations. It results in an increase in weight values proportional to the corresponding positive input values on the preceding steps. Note, in the AHC-algorithm, $y_i(t-1,t-1)$ is a trace of previous values of y_i, called the eligibility trace. Thus, the above expression is a simplification to investigate learning in hidden units rather than speed up learning.
Then, the author considered using the error backpropagation to the hidden units of the evaluation network. Note, σ_t plays the role of an error in the update of the weights of the output unit. Thus, the error of the output unit, denoted δ_m^p, is given by

$$\delta_m^p(t-1) = \sigma_t$$

where the superscript denotes the association with the evaluation network that generates output p. Hence, the error backpropagated from the output unit to the Hidden Unit j is just σ_t. Thus, the error of Hidden Unit j, denoted δ_j^p becomes

$$\delta_j^p(t-1) = \delta_m^p(t-1) g\big(v_{j+n,m,t-1}\big) y_j(t-1,t-1)\big(1 - y_j(t-1,t-1)\big)$$

where g is the sign function. We update the weights of the hidden units as follows:

$$v_{i,j,t} = v_{i,j,t-1} + \beta_h \delta_j^p(t-1) x_{i,t-1} + \beta_m \Delta v_{i,j,t-1}$$

for units $j \in H$ and inputs $i = 0,...,n$.

16.2.2.0.2 The action network The hidden unit outputs a_j are given by

$$a_j(t) = f\big(\sum_{i=0}^{n} x_{i,t} w_{i,j}\big) \text{ for } j \in H$$

They partly determine the input vector z for the output units, along with the input from the environment:

$$\begin{aligned} z_i(t) &= x_{i,t} \text{ for } i = 0,...,n \\ z_i(t) &= a_{i-n}(t) \text{ for } i = n+1,...,n+m_h \end{aligned}$$

Let $L_t \subset O$ be the set of indices for the output units representing legal actions for the state at time t. Let s_j be the noisy weighted sum of the input for Unit j, $j \in L_t$ given by

$$s_j(t) = \sum_{i=0}^{n+m_h} z_i(t) w_{i,j,t} + \nu_j(t)$$

where $\nu_j(t)$ is a sequence of random variables from a probability distribution. The unit with the largest value for s_j wins the competition and is assigned a nonzero output:

$$a_j(t) = \begin{cases} 1 \text{ if } s_j(t) > s_k(t) \text{ for } k \in L_t \text{ and } j \neq k \\ 0 \text{ otherwise} \end{cases}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The Output Unit j, $j \in L_t$, updates its weights as follows:

$$w_{i,j,t} = w_{i,j,t-1} + \rho\sigma_t\big(a_j(t-1) - E[a_j(t-1)|\, w, z]\big)z_i(t-1)$$

for $i = 0, ..., n + m_h$, where $E[a_j(t-1)|\, w, z]$ is the expected value of $a_j(t-1)$ given w and z.
We now consider updating the weights of the hidden units. The error in the output of Output Unit k, $k \in L_t$, is given by

$$\delta_k^a = \sigma_t\big(a_k(t-1) - E[a_k(t-1)|\, w, z]\big)$$

The backpropagated error to Hidden Unit j is used to compute the error of the hidden unit:

$$\delta_j^a(t-1) = \sum_{k \in L_t} \big(\delta_k^a(t-1)g(w_{j+n,k,t-1})\big)z_j(t-1)\big(1 - z_j(t-1)\big)$$

and the weights are updated as follows:

$$w_{i,j,t} = w_{i,j,t-1} + \rho_h\delta_j^a(t-1)x_{i,t-1} + \rho_m\Delta w_{i,j,t-1}$$

for units $j \in H$ and inputs $i = 0, ..., n$.

16.2.3 Implementing the TDBP model

In order to present the implementation of the $TD(\lambda)$-procedures and the TDBP model, we follow the notation in Section (5.3), where we described backpropagation for a multilayer neural network.

16.2.3.1 Feedforward networks

We consider a multi-layer feedforward network (see Section (5.3)) and assume a set of artificial neurons (or units) $\{f_{i,j,k}\}_{k=0}^K$, where the multilayer subscript $k = 0$ corresponds to the set of inputs $\{x_i\}_{i=1}^{N_i}$, and the remaining subscript k corresponds to the set of outputs $\{y_i\}_{i=1}^{N_{o,k}}$, where $N_{o,k}$ is the number of output in the k-th layer.
In that setting, we define the output function of the jth node for the k-th layer as

$$O_{j,k}(x, w) = g(net_{j,k}(x, w))\,,\, j = 1, .., N_{o,k}$$

where $g(\cdot)$ is a differentiable function. We let the corresponding net function satisfies

$$net_{j,k}(x, w) = \sum_{i=1}^{N_{o,k-1}} O_{i,k-1}(x, w)w_{i,j}^{k-1} + b_{j,k} \text{ and } O_{j,0}(x, w) = \sum_{i=1}^{N_i} x_i$$

where $w_{i,j}^{k-1}$ is the weight going from the ith node in layer $k-1$ to the jth node in layer k. The error function for the sth neuron on the Kth-layer satisfies

$$E_s(x, w, d) = \big(O_{s,K}(x, w) - d_s\big)^2\,,\, s = 1, ..., N_{o,K}$$

where d_s is the sth element of the target vector d. The total error of the network is the sum of the errors of all the neurons in the output layer

$$E(x, w, d) = \frac{1}{2}\|O(x, w) - d\|^2 = \frac{1}{2}\sum_{s=1}^{N_{o,K}} E_s(x, w, d)$$

The gradient of the total error with respect to the weight $w_{i,j}^{k-1}$ is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}\nabla E(w_{i,j}^{k-1}) = \frac{\partial}{\partial w_{i,j}^{k-1}} E(x,w,d) &= \frac{\partial E(x,w,d)}{\partial net_{j,k}} \frac{\partial net_{j,k}}{\partial w_{i,j}^{k-1}} \\ &= \frac{\partial E(x,w,d)}{\partial O_{j,k}(x,w)} \frac{\partial O_{j,k}(x,w)}{\partial net_{j,k}} \frac{\partial net_{j,k}}{\partial w_{i,j}^{k-1}}\end{aligned}$$

We let $e_{j,k} = \frac{\partial E(x,w,d)}{\partial O_{j,k}(x,w)}$ be the pre-error signal for the jth unit on the kth-layer, and

$$\delta_{j,k} = \frac{\partial E(x,w,d)}{\partial net_{j,k}} = e_{j,k} \frac{\partial O_{j,k}(x,w)}{\partial net_{j,k}}$$

be the error signal. The gradient of the total error (see Equation (5.3.13)) for the weight $w_{i,j}^{k-1}$ is

$$\begin{aligned}\nabla E(w_{i,j}^{k-1}) &= \delta_{j,k} O_{i,k-1}(x,w) \\ &= e_{j,k} \frac{\partial O_{j,k}(x,w)}{\partial net_{j,k}} O_{i,k-1}(x,w)\end{aligned}$$

with stochastic gradient descent rule given by

$$\Delta w_{i,j}^{k-1} = -\eta \nabla E(w_{i,j}^{k-1}) \tag{16.2.23}$$

In the output layer K, the pre-error signal is

$$e_{j,K} = \frac{\partial}{\partial O_{j,K}} E = (O_{j,K} - d_j)$$

which is the known error between the output function and the target value.
In the layer $K-l$, for a given $l \in [1,K]$, the pre-error signal is

$$e_{j,K-l} = \sum_{t=1}^{N_{o,K-l+1}} \delta_{t,K-l+1} w_{j,t}^{k-l}$$

and the error signal is

$$\delta_{j,K-l} = e_{j,K-l} \frac{\partial O_{j,K-l}(x,w)}{\partial net_{j,K-l}}$$

16.2.3.2 Implementing the TD-procedures

We present the implementation of the $TD(\lambda)$-procedures proposed by Sutton [1989] in the case where the outcome z of the value function in Equation (16.1.8) is a vector. Again, the vector z only depends on the terminal state q_{m+1}, that is, z is $\mathcal{F}_T$-measurable. Note, there is no notion of multiple hidden layers in Sutton's implementation. Still, we assume multiple layers in our notation as it will be used later in the implementation of the TDBP model. We focus on the implementation of the $TD(0)$-procedures, and delay the presentation of the implementation of the $TD(\lambda)$-procedures to the implementation of the TDBP model.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

16.2.3.2.1 The network We consider a multi-layer feedforward network (see Section (5.2)) with implementation defined above. As discussed in Section (16.1.3), the TD-procedures introduce the notion of time, resulting in a sequence of feedforward networks to represent the prediction functions. At time t, in the output layer K, the set of outputs is $\{y_i^t\}_{i=1}^{N_{o,K}}$ and the output function of the jth node is $O_{j,K}^t$. For $s = 1, ..., N_{o,K}$, the output y_s^t corresponds to the prediction function P_s^t, and z_s is the component of the outcome vector corresponding to the output unit s. Thus, in the TD-procedures, z_s is predicted by each P_s^t, $t = 1, ..., m$, where m is the number of observation vectors $x_1, x_2, ..., x_m$ presented before the outcome z_s. By definition, we have

$$P_s^{m+1} = z_s \text{ , } s = 1, ..., N_{o,K}$$

We let $w_{i,j,k-1}^t$ be the weight at time t going from the ith node in layer $k-1$ to the jth node in layer k. The corresponding net function satisfies

$$net_{j,k}^t(x,w) = \sum_{i=1}^{N_{o,k-1}} O_{i,k-1}^t(x,w) w_{i,j,k-1}^t + b_{j,k}^t \text{ and } O_{j,0}^t(x,w) = \sum_{i=1}^{N_i} x_i$$

In that setting, the output function of the jth node for the k-th layer is

$$O_{j,k}^t(x,w) = g(net_{j,k}^t(x,w)) \text{ , } j = 1, .., N_{o,k}$$

where $g(\cdot)$ is a differentiable function.

16.2.3.2.2 The $TD(0)$ case Note, in the $TD(\lambda)$ procedure with update rule given in Equation (16.1.12), there is no total error that we need to differentiate with respect to the network weights since there is a single output unit. Hence, the TD error is

$$\sigma_{t+1} = P_{t+1} - P_t$$

which is known at time $t+1$. However, in the case where z is a vector, the error function at time t is given by

$$E_t = \frac{1}{2} \sum_{s=1}^{N_{o,K}} \left(P_s^{t+1} - P_s^t\right)^2 = \frac{1}{2} \sum_{s=1}^{N_{o,K}} \left(O_{s,K}^{t+1} - O_{s,K}^t\right)^2$$

The target value is associated to the prediction function P_s^{t+1}, so that the update rule only accounts for the effects of the weights on the earlier predictions P_s^t, not the later one P_s^{t+1}. Hence, the update rule in Equation (16.2.23), for the weight $w_{i,j,k-1}^t$, is given by

$$\Delta w_{i,j,k-1}^t = \alpha \nabla E(w_{i,j,k-1}^t)$$

since we have swapped the position of the target value. Following the above section, we can rewrite the target rule for multi-layer feedforward networks as follows:

$$\begin{aligned} w_{i,j,k-1}^{t+1} &= w_{i,j,k-1}^t + \alpha \delta_{j,k}^t O_{i,k-1}^t(x,w) \\ &= w_{i,j,k-1}^t + \alpha e_{j,k}^t \frac{\partial O_{j,k}^t(x,w)}{\partial net_{j,k}^t} O_{i,k-1}^t(x,w) \end{aligned} \tag{16.2.24}$$

For ease of notation, we only detail the case of a network with a single hidden layer. In the output layer K, we go from the ith node in layer $(K-1)$ to the jth node in layer K. The pre-error signal is

$$e_{j,K}^t = \frac{\partial}{\partial O_{j,K}} E = \left(O_{j,K}^{t+1} - O_{j,K}^t\right)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

and the update rule is

$$w_{i,j,k-1}^{t+1} = w_{i,j,k-1}^{t} + \alpha\big(O_{j,K}^{t+1} - O_{j,K}^{t}\big)\frac{\partial O_{j,K}^{t}(x,w)}{\partial net_{j,K}^{t}} O_{i,K-1}^{t}(x,w)$$

***Remark* 16.2.1** *This update rule is*

$$w_{i,j,K-1}^{t+1} = w_{i,j,K-1}^{t} + \alpha\big(O_{j,K}^{t+1} - O_{j,K}^{t}\big)\frac{\partial O_{j,K}^{t}(x,w)}{w_{i,j,K-1}^{t}}$$

which is the update rule of the $TD(0)$-procedure in Equation (16.1.13). It only applies to a two-layer network made of input-output units.

The next step is to consider the hidden layer $(K-1)$ and compute the gradient for the weight $w_{i,j,k-2}$ going from the ith node on layer $(K-2)$ to the jth node in layer $(K-1)$. For notation purpose, we use the subscript s to represent the nodes on the Kth-layer. The gradient $\nabla E(w_{i,j,k-2})$ for the weight $w_{i,j,k-2}$ becomes

$$\nabla E(w_{i,j,k-2}^{t}) = \frac{\partial}{\partial w_{i,j,k-2}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} \big(O_{s,K}^{t+1} - O_{s,K}^{t}\big)\frac{\partial}{\partial w_{i,j,k-2}^{t}}\big(O_{s,K}^{t+1} - O_{s,K}^{t}\big)$$

The pre-error signal is

$$e_{j,K-1}^{t} = \frac{\partial}{\partial O_{j,K-1}^{t}} E = \sum_{s=1}^{N_{o,K}} \delta_{s,K} w_{j,s,k-1}^{t}$$

The error signal is

$$\delta_{j,K-1}^{t} = e_{j,K-1}^{t}\frac{\partial O_{j,K-1}(x,w)}{\partial net_{j,K-1}}$$

and the gradient simplifies to

$$\nabla E(w_{i,j,k-2}^{t}) = \delta_{j,K-1}^{t} O_{i,K-2}^{t}(x,w)$$

We can then compute the update rule in Equation (16.2.24).

16.2.3.3 The TDBP model

As stated by Sutton [1988], the TD methods are applicable for predictions $P(x_t, w)$ formed by nonlinear functions. The key requirement being that the gradient $\nabla_w P_t$ is computable, which is the case with backpropagation models. Sutton [1989] described the implementation of the $TD(\lambda)$-procedures in the case where the output z is a vector. McClelland [2015] further extended this approach to the case of multi-layer feedforward networks, and proposed a simple implementation of the temporal difference backpropagation (TDBP) model. Following this approach, we extend the implementation of the $TD(\lambda)$-procedures described in the above section to the case where the feedforward network for the prediction function is a Deep Neural Network.

In the temporal difference method, the change in weight for the $TD(\lambda)$ procedure is given in Equation (16.1.12) by

$$\Delta w_t = \alpha\sigma_{t+1}\sum_{n=1}^{t}\lambda^{t-n}\nabla_w P_n$$

where the trace is denoted by $e_t = \sum_{n=1}^{t}\lambda^{t-n}\nabla_w P_n$ and the TD error is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\sigma_{t+1} = P_{t+1} - P_t$$

The trace (or eligibility term) performs part of the credit assignment by determining which weights are eligible for a specific modification in the presence of a TD error. As discussed above, when the outcome z is a vector, the update rule for the weight $w_{i,j}^t$ becomes

$$\Delta w_{i,j}^t = \alpha \sum_{s=1}^{N_{o,K}} \sigma_{t+1}(s) e_{i,j,s}^t \tag{16.2.25}$$

with trace $e_{i,j,s}^t = \sum_{n=1}^t \lambda^{t-n} \nabla_{w_{i,j}^n} P_s^n$ and TD error given by

$$\sigma_{t+1}(s) = P_s^{t+1} - P_s^t$$

As discussed in Section (16.1.2), the TD error for the sth output unit is only known at time $t+1$. Hence, we can only compute the update rule at time $t+1$. To do so, we compute the gradient for the predicting function P_t at time t and save it until the TD error σ_{t+1} is known at time $t+1$. Note, this is computationally intensive.
In our setting, the update rule for the weight $w_{i,j,k-1}^t$, with node i on the $k-1$ layer and node j on the k layer, is given by

$$w_{i,j,k-1}^{t+1} = w_{i,j,k-1}^t + \alpha \sum_{s=1}^{N_{o,K}} \left(O_{s,K}^{t+1}(x,w) - O_{s,K}^t(x,w)\right) e_{i,j,s}^t$$

where $e_{i,j,s}^t$ is the s-th eligibility at time t for the weight going from node i at layer $k-1$ to node j at layer k given by

$$e_{i,j,s}^t = \sum_{n=1}^t \lambda^{t-n} \nabla_{w_{i,j}^n} O_{s,K}^n(x,w)$$

which is computed as follows:

$$e_{i,j,s}^t = \lambda e_{i,j,s}^{t-1} + \frac{\partial O_{s,K}(x,w)_t}{\partial w_{i,j,k-1}^t}$$

Note, compared with classical backpropagation models, there is no total error that we need to differentiate with respect to the network weights. Thus, there is no pre-error signal involved. At time t, our neural network has the output functions $P_s^t = O_{s,K}(x,w)_t$, for $s = 1, ..., N_{o,K}$. Hence, for each time $n = 1, ..., t$, we need to compute the derivatives of the output function $O_{s,K}^n(x,w)$, $s = 1, .., N_{o,K}$, with respect to the weight $w_{i,j,k-1}^n$ going from the ith node on layer $k-1$ to the jth node in layer k. That is, we are only interested in computing the derivatives

$$\frac{\partial O_{s,K}(x,w)_t}{\partial w_{i,j,k-1}^t}, s = 1, ..., N_{o,K}$$

Using the chain rule, we get

$$\frac{\partial O_{s,K}(x,w)_t}{\partial w_{i,j,k-1}^t} = \frac{\partial O_{s,K}(x,w)_t}{\partial net_{j,k}} \frac{\partial net_{j,k}}{\partial w_{i,j,k-1}^t}$$

We let the error signal for the jth unit on the k-th layer be

$$\begin{aligned} \delta_{j,k}^t &= \frac{\partial O_{s,K}(x,w)_t}{\partial net_{j,k}} \\ &= \frac{\partial O_{s,K}(x,w)_t}{\partial net_{l,K}} \frac{\partial net_{l,K}}{\partial O_{m,K-1}(x,w)_t} \cdots \frac{\partial O_{j,k}(x,w)_t}{\partial net_{j,k}} \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where l is the subscript for the nodes on the K layer and m is the subscript for the nodes on the $K-1$ layer. Note, all the error signal $\delta^t_{j,k}$ are stored in a 2D-matrix of size $N_{o,k} \times N_{o,K}$. At the output layer K, the error signal $\delta_{.,K}$ is a vector of length $N_{o,K}$. The gradients of the output units rewrite as

$$\frac{\partial O_{s,K}(x,w)_t}{\partial w^t_{i,j,k-1}} = \delta^t_{j,k} O_{i,k-1}(x,w)_t \ , \ s = 1, ..., N_{o,K}$$

All these gradients are stored in a 3D-matrix of size $N_{o,k-1} \times N_{o,k} \times N_{o,K}$. We can then scale the gradients $\frac{\partial O_{s,K}(x,w)_t}{\partial w^{k-1}_{i,j}}$ with the parameter λ and compute the trace. The trace at time t, for the output node s, is a 3D-matrix denoted $e^t_{i,j}(s)$ and given by

$$e^t_{i,j}(s) = \sum_{n=1}^{t} \lambda^{t-n} \nabla_{w^n_{i,j}} P_n(s) \ , \ P_n(s) = O_{s,K}(x,w)_n$$

Knowing the trace at time t, for the output node s, we can then compute the update rule in Equation (16.2.25).

16.3 Beyond TD-learning

16.3.1 State representation

Tsitsiklis et al. [1997], Sutton et al. [2009], among others, showed that TD-learning was diverging for non-linear function approximator. The reason being that TD-learning is not a true gradient descent algorithm as it ignores the derivative of the TD target. There are several methods for learning generative models of dynamical systems, among which are the predictive representations (PR). The main idea of PR is that knowledge about the world can be represented in terms of predictions of relationships between the observable quantities. Rivest et al. [1990], Jaeger [1998], Littman et al. [2002] proposed different methods for learning a predictive representation of state. Tanner et al. [2005a] introduced the TD-networks as a formalism for expressing and learning knowledge in a predictive form. Maei et al. [2009] proposed the gradient temporal-difference (GTD) learning, which minimises an objective function corresponding to the error in the Bellman equation. Silver [2012] combined the GTD with backpropagation through time (BPTT), or, real-time recurrent learning (see Williams et al. [1989]), obtaining a generalisation of the Bellman equation corresponding to the semantics of TD networks.

16.3.1.1 Predictive state representation

Littman et al. [2002] proposed the predictive state representations (PSRs), where each prediction is an estimate of the probability of some sequence of observations given a sequence of actions. TD-networks is a generalisation of PSRs, where each prediction is an estimate of the probability (or expected value) of some function of future predictions and observations. The predictions (of future observations) are "answers" to a set of "questions" represented within the network.

16.3.1.1.1 Description We consider an agent receiving a time series of m-dimensional observations o_t at time-step t and taking action a_t, where $o_t \in \mathcal{O}$ and $a_t \in \mathcal{A}$. The action and observation events occur in a sequence $a_{t-1}, o_t, a_t, o_{t+1}, ...$, called experience. We focus on predicting action-conditional functions of future experience in partially observable environments.

Notation 16.3.1 *A history h_t is a length t sequence of observations $h_t = o_1, ..., o_t$. It is assumed that histories are generated by a Markov chain with a probability $1-\gamma$ of terminating after every transition, to give a history of length τ. The Markov chain has unknown transition probabilities $T_{h,ho} = \gamma P(o_{t+1} = o | \ h_t = h, \tau > t)$, with zero probability for all other transitions. This Markov chain has a well-defined distribution $d(h)$ over the set $\mathcal{H}$ of all histories $d(h_t) = P(\tau = t, \ o_1, ..., o_t = h_t)$.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

A state representation is a function $v(h)$ mapping histories to a vector of state variables $v_i(h)$. For a state representation to be of use online, it must be incremental. That is, the state can be updated from one observation to the next:

$$v(h_{t+1}) = f(v(h_t), o_{t+1})$$

A predictive state representation (PSR) is a state representation in which all state variables are predictions of future events. The target (or payoff) of a prediction is a function of future observations:

$$z_t = g\big(o_{t+1}, o_{t+2}, ...\big) \tag{16.3.26}$$

The true answer for a prediction is the expected value of the target, given a history of observations h:

$$v^*(h) = E[z_t|\, h_t = h] \tag{16.3.27}$$

The PSR learns an estimated answer $v(h, w) \approx v^*(h)$ to each prediction, by adjusting the parameters w. The answer vector $v(h_t, w)$ is the state representation used by the agent at time-step t.
A temporal-difference (TD) network consists of a question network (prediction) and an answer network (following prediction) (see Section (16.1.3.5)). The former defines the semantics of the compositional predictions: how a target depends on subsequent targets. In the case of linear question networks, the target is a linear combination of the subsequent target and observations:

$$z_t = Po_{t+1} + Qz_{t+1} \tag{16.3.28}$$

Linear question networks satisfy the Bellman network equation (properties of conditional expectation)

$$\begin{aligned} v^*(h_t) &= E[z_t|\, h_t] \\ &= E[Po_{t+1} + Qz_{t+1}|\, h_t] \\ &= E[Po_{t+1} + Qv^*(h_{t+1})|\, h_t] \end{aligned}$$

As an example of a simple prediction, the target is the observation at the next time-step ($P = 1$ and $Q = 0$):

$$v_1^*(h) = E[o_{t+1}|\, h_t = h]$$

As an example of a compositional prediction, it can be expressed as the expected value at time-step $t + 1$ of the expected observation at time-step $t + 2$:

$$v_2^*(h) = E[v_1^*(h_{t+1})|\, h_t = h]$$

Compositional questions can be recursive, so that questions can be asked about temporally distant observations. For instance, a value function can be represented by the prediction ($P = 0$ and $Q = 1$)

$$v_3^*(h) = E[o_t + v_3^*(h_{t+1})|\, h_t = h]$$

where o_t can be a reward and v_3^* is the value function for that reward.
The answer network is a non-linear representation of state containing n state variables, where each state variable $v_i(h)$ represents the estimated answer to the ith prediction. The answers are updated incrementally by combining the state $v(h_{t-1})$ at the last time-step, that is, the previous answers, with the new observation o_t. In the case where the answers are represented by recurrent neural network (RNN), we get

$$v(h_t, w) = g\big(Ao_t + Bv(h_{t-1})\big)$$

where $g(x)$ is a non-linear activation function with derivative $g^{'}(x)$, $w = [A, B]$ is an $n \times (m + n)$ weight matrix, and $v(h_0) = 0$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

16.3.1.1.2 The TD networks In the TD model introduced by Tanner et al. [2005a], we get a network of nodes, each representing a single scalar prediction. The links between nodes and/or observations represent the target for learning the prediction. The nodes correspond to questions about the system, so that the nodes and the links form the question network. The authors assumed each observation to be a single bit 0 or 1 and presented a network where the question asked by the node y^1 at time t is: if the next action is $a1$, what is the probability for the next observation o_{t+1} to be 1? Similarly, the question for node y^4 at time t is: if the next action is $a2$, what is the expected value of the prediction from node y^1 at time $t+1$? See details in Figure (16.10).

Figure 16.10: Symmetric TD network

We denote the prediction for node i at time step t by $y_t^i \in [0,1]$, $i = 1, ..., n$. The column vector of predictions $y_t = \left(y_t^1, ..., y_t^n\right)^\top$ is updated by the rule

$$y_t = \sigma(W_t x_t)$$

where $x_t \in \mathbb{R}^m$ is a feature vector given by

$$x_t = x(a_{t-1}, o_t, y_{t-1})$$

W_t is an $n \times m$ matrix of weights, and σ is the logistic function. The five steps of the TD network are as follows:

1. Choose an action a_{t-1} and receive an observation o_t.
2. Calculate the input vector x_t.
3. Create the new predictions $y_t = \sigma(W_t x_t)$
4. Calculate the targets z_{t-1} from the previous predictions y_{t-1} using y_t and the observation o_t according to the links of the question network and the action conditions.
5. Update the weights W according to $(z_{t-1} - y_{t-1})$.

A network where each node i has a single target is called a single-target question network. The target node i is the parent of i denoted $p(i)$. The later parents of node i : $\{p(p(i)), p(p(p(i))), ...\}$ are denoted $\{p^2(i), p^3(i), ...\}$. Then, the target z^i for node i is

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$z_t^i = \begin{cases} y_{t+1}^{p(i)} \text{ or } o_{t+1} \text{ if } c_t^i = 1 \\ y_t^i \text{ if } c_t^i = 0 \end{cases}$$

where $c_t^i \in \{0,1\}$ indicates if the action condition a_t for y^i was met, or not, at time t. Each component w_t^{ij} of W_t is updated by the learning rule

$$w_{t+1}^{ij} = w_t^{ij} + \alpha\big(z_t^i - y_t^i\big)\frac{\partial y_t^i}{\partial w_t^{ij}}$$

Note, if $c_t^i = 0$, then the error term $\big(z_t^i - y_t^i\big)$ is zero. Given the update equation above, the rule simplifies to

$$\Delta w_t^{ij} = \alpha\big(z_t^i - y_t^i\big)y_t^i\big(1 - y_t^i\big)x_t^j$$

16.3.1.1.3 The Bellman network equation In the TD model, Silver [2012] showed that we train the parameters of the answer network, so that the estimated answers approximate the true answers as closely as possible. The quality of the answers can be measured by using the Bellman network equation. Seek answers $v(h,w)$ approximately satisfying this equation. It can be solved by bootstrapping, that is, by using the RHS of $Po_{t+1} + Qv(h_{t+1},w)$ as a surrogate for the true value function $v^*(h_t)$. The parameters of the answer network can then be adjusted online by gradient descent by minimising the mean squared error

$$MSE(w) = E[\sum_{i=1}^{n}\big(v_i^*(h_t) - v_i(h_t,w)\big)^2]$$

with gradient

$$\begin{aligned} -\frac{1}{2}\nabla_w MSE(w) &= E[\sum_{i=1}^{n}\big(v_i^*(h_t) - v_i(h_t,w)\big)\nabla_w v_i(h_t,w)] \\ &= E[\phi(h_t,w)\big(v^*(h_t) - v(h_t,w)\big)] \end{aligned}$$

where $\phi(h,w)$ is a matrix combining the gradients for each answer

$$\phi(h,w) = \big[\nabla_w v_1(h,w), ..., \nabla_w v_n(h,w)\big]$$

Sampling the gradient gives a stochastic gradient descent algorithm

$$\begin{aligned} \Delta w &= \alpha\phi(h_t,w)\big(v^*(h_t) - v(h_t,w)\big) \\ &\approx \alpha\phi(h_t,w)\big(Po_{t+1} + Qv(h_{t+1},w) - v(h_t,w)\big) \\ &= \alpha\phi(h_t,w)\sigma(h_{t+1},w) \end{aligned}$$

where $\sigma(h_{t+1},w)$ is the TD error

$$\sigma(h_{t+1},w) = Po_{t+1} + Qv(h_{t+1},w) - v(h_t,w)$$

Note, Sutton et al. did not follow the full recurrent gradient, but a computationally expedient one-step approximation given by

$$\nabla_w v_i(h_t) = \nabla_w g\big(A_{i,\cdot}o(h_t) + B_{i,\cdot}v(h_{t-1})\big)_i$$

with

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\frac{\partial v_i(h_t)}{\partial A_{j,k}} \approx \begin{cases} o_k(h_t)g^{'}(v_i(h_t)) \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}$$

$$\Delta A_{i,k} = \alpha\sigma_i(h_{t+1}, w)o_k(h_t)g^{'}(v_i(h_t))$$

and

$$\frac{\partial v_i(h_t)}{\partial B_{j,k}} \approx \begin{cases} v_k(h_{t-1})g^{'}(v_i(h_t)) \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}$$

$$\Delta B_{i,k} = \alpha\sigma_i(h_{t+1}, w)v_k(h_{t-1})g^{'}(v_i(h_t))$$

The simple TD network learning rule only considers direct connections from previous state variables $v(h_{t-1})$ to current state variables $v(h_t)$, ignoring the past state variables. The model was extended to incorporate histories (see Tanner et al. [2005a]) and to learn over longer time scales by using a $TD(\lambda)$-like algorithm (see Tanner et al. [2005b]).

16.3.1.2 Gradient temporal difference networks

We follow the notation from Section (15.3.2), where $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, the value function is $V_s = E[R_t|\, s_t = s]$, $R(s) = E[r_{t+1}|\, s_t = s]$, and $T_{ss'} = P(s^{'}|s)$ is an $n \times n$ transition matrix. In the case where the action a is not a random variable, the Bellman equation in Equation (15.3.7) is given by

$$V_s = R_s + \gamma T_{ss'} V_s$$

The true value function V_s is computed with the function approximator $V_w(s) = f(s, w)$ with parameters w updated by stochastic gradient descent, so as to minimise the mean squared error between the value function and the return:

$$\Delta w_t = \alpha\big(R_t - V_w(s_t)\big)\nabla_w V_w(s_t)$$

where α is the learning rate. The return R_t is unknown at time t and has a high variance. The temporal difference (TD) replace the return R_t in the update equation with a lower variance, one-step estimate value $r_t + \gamma V_w(s_{t+1})$, called the TD target. The TD-learning update is given by

$$\Delta w_t = \alpha\sigma_{t+1}\nabla_w V_w(s_t)$$

where the TD error is

$$\sigma_{t+1} = r_t + \gamma V_w(s_{t+1}) - V_w(s_t)$$

We saw in Section (16.1.3) that TD-learning was effective with a linear function approximator $V_w(s) = \phi(s) \cdot w$. However, for non-linear function approximator, Tsitsiklis et al. [1997], Sutton et al. [2009] showed that TD-learning was diverging. The reason being that TD-learning is not a true gradient descent algorithm as it ignores the derivative of the TD target.
Maei et al. [2009] proposed the gradient temporal-difference (GTD) learning, which minimises an objective function corresponding to the error in the Bellman equation. It measures the error between the value function $V_w(s)$ and the TD target given by the Bellman equation

$$V_w(s) = R_s + \gamma T_{ss'} V_w(s)$$

Silver [2012] combined the GTD with backpropagation through time (BPTT) or real-time recurrent learning (see Williams et al. [1989]), obtaining a generalisation of the Bellman equation corresponding to the semantics of TD networks. That is, the Bellman equation applies to linear question networks. Since the Bellman target equation lies

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

outside the space of value functions represented by the function approximator f, the author projected it back into this space by using a projection Π_w, leading to the mean squared projected Bellman error (MSPBE)

$$J(w) = \|V_w - \Pi_w\big(R + \gamma T V_w\big)\|_\rho^2$$

where $\rho(s)$ is the stationary distribution of the Markov reward process. To account for tractability, the projection operator Π_w is a linear projection onto the tangent space of V_w. That is,

$$\Pi_w = \Phi_w\big(\Phi_w^\top D \Phi_w\big)^{-1}\Phi_w^\top D$$

where D is the diagonal matrix $D = diag(\rho)$ and Φ_w is the tangent space of V_w, where each row is a value gradient $\phi(s) = (\Phi_w)_{s,\cdot} = \nabla_x V_w(s)$.
The GTD2 algorithm minimises the MSPBE by stochastic gradient descent. The MSPBE can be expressed as the product of three expectations:

$$J(w) = E[\phi(s)\sigma]^\top E[\phi(s)\phi^\top(s)]^{-1} E[\phi(s)\sigma]$$

which can be written as

$$J(w) = E[\phi(s)\sigma]^\top \psi_w$$

where $\psi_w = E[\phi(s)\phi^\top(s)]^{-1}E[\phi(s)\sigma]$. The algorithm estimates both

- $\psi \approx \psi_w$ by stochastic gradient descent, and
- minimises $J(w)$ by stochastic gradient, assuming that the current estimate is correct, that is, $\psi \approx \psi_w$.

As a result, the update applied at every time step t with step size parameters α and γ is given by

$$\begin{aligned}
\xi_t &= \big(\sigma_{t+1} - \phi(s_t)\cdot\psi\big)\nabla_w^2 V(s)\psi \\
\Delta w_t &= \alpha\big(\phi(s_t) - \gamma\phi(s_{t+1})(\phi(s_t)\cdot\psi)\big) - \xi_t \\
\Delta\psi_t &= \beta\phi(s_t)\big(\sigma_{t+1} - \phi(s_t)\cdot\psi\big)
\end{aligned}$$

This algorithm converges to a local minimum of $J(w)$ even the function approximator f is non-linear (see Maei et al. [2009], Sutton et al. [2009]). Note, the TDC algorithm minimises the same objective but with a sightly different update equation:

$$\Delta w_t = \alpha\big(\phi(s_t)\sigma - \gamma\phi(s_{t+1})(\phi(s_t)\cdot\psi)\big) - \xi_t$$

16.3.2 A link to forward start contract

We described the TD methods in Section (16.1.3), and saw in Section (16.1.3.5) that they could be viewed as gradient descent in the space of the modifiable parameters. We briefly introduced predictive state representations (PSRs) in Section (16.3.1), and hinted at the TD-networks, where the predictions of future observations are "answers" to a set of "questions" represented within the network. Since from Remark (16.1.7) the real goal of the TD-procedures is for each prediction (question) to match the expected value of the subsequent outcome (answer), it is a special case of PSR. We are going to show that, for an agent, TD learning is about evaluating a forward start contract, a well known product in finance, and rolling his position in time in a question-answer relation.
We consider the Ito process $S_t = f(X_t)$, $t \geqslant 0$, where X is a state vector, and f is a differentiable function. We denote by t_0 today's date, and t_1 and t_2 two future dates with $t_2 > t_1 > t_0$. The conditional probability density function (*pdf*), at time t_1, for the random variable S_{t_2}, is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\phi(S_{t_2} \in dx \,|\mathcal{F}_{t_1})$$

where the filtration $\mathcal{F}_{t_1}$ can be considered as all available information at time t_1. The knowledge of this function is of fundamental importance if we want to compute forward start payoffs. We denote $C^F(t_0, t_1; T)$ the value at time t_0 of a forward start payoff, $g(S_{t_1}, S_T)$, starting at time t_1 with maturity T. Its value is given by

$$C^F(t_0, t_1; T) = \mathbb{E}\left[g(S_{t_1}, S_T)|\,\mathcal{F}_{t_0}\right]$$

where $g(\cdot,\cdot)$ is a function. By conditioning the forward start contract with information available at time t_1, *i.e.* by the filtration $\mathcal{F}_{t_1}$, we get

$$\begin{aligned} C^F(t_0, t_1; T) &= \mathbb{E}\left[g(S_{t_1}, S_T)\,|\,\mathcal{F}_{t_0}\right] \\ &= \mathbb{E}\left[\mathbb{E}\left[g(S_{t_1}, S_T)\,|\,\mathcal{F}_{t_1}\right]|\,\mathcal{F}_{t_0}\right] \end{aligned}$$

For any set $\mathcal{S}$, the probability that $X_{t_1} \in \mathcal{S}$ at t_1, given X_{t_0} at t_0, is

$$P(X_{t_1} \in \mathcal{S}|X_{t_0}) = \int_{\mathcal{S}} \phi(x, t_1, X_{t_0}, t_0)\lambda(dx)$$

where λ is the Lebesgue measure. Thus, the value of the forward start contract, seen at time t_0, can be rewritten as

$$C^F(t_0, t_1; T) = \int \mathbb{E}\left[g(S_{t_1}, S_T)\,|\,X_{t_1} = x_{t_1}\right]\phi(x_{t_1}, t_1, X_{t_0}, t_0)\lambda(dx_{t_1}) \tag{16.3.29}$$

where the integrand is

$$\mathbb{E}\left[g(S_{t_1}, S_T)\,|\,X_{t_1} = x_{t_1}\right] = \int g(S_{t_1}, S_T)\phi(x_T, T, x_{t_1}, t_1)\lambda(dx_T) \tag{16.3.30}$$

Definition 16.3.1 (Kolmogorov-Compatibility) *Any future deterministic conditional density, such that*

$$\phi(X_0; T - t_0) = \int \phi(X_0; t_1 - t_0)\phi(x_{t_1}; T - t_1)d\lambda(x_{t_1}) \tag{16.3.31}$$

is satisfied, defines a Kolmogorov-compatible density.

There is in general an infinity of solutions for the forward density

$$\phi(x_{t_1}; T - t_1)$$

such that Definition (16.3.1) is satisfied.
Being at time t_0, Equation (16.3.30) is the forward start payoff starting at time t_1 with maturity T. Similarly, given $t_0 < t_1 < t_2 < T$, and using conditional expectation with respect to time, we can rewrite the expected value at time t_0 as

$$C^F(t_0, t_1, t_2; T) = E[E[E[z_T|\,X_{t_2} = x_{t_2}]|\,X_{t_1} = x_{t_1}]|\,X_{t_0} = x_{t_0}]$$

with $z = g(S_T)$. We can apply the same principle to any discrete time $t_0 < t_1 < \cdots < T$.
In the TD-model described in Section (16.1.3), the prediction function $P_t = P(x_t, w)$ is meant to approximate $E[z|\,x_t]$. However, when expressing TD-methods as gradient descent in Section (16.1.3.5), the term $E[z|\,x_t]$ is approximated by $P_{t+1} = P(x_{t+1}, w)$ in the TD error

$$\sigma_t = \gamma P_{t+1} - P_t$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

which is the immediately following prediction. That is, we want the prediction P_t to match the subsequent outcome P_{t+1}. We observe that P_{t+1} corresponds to the integrand in Equation (16.3.30) of a forward start contract seen at time t. At time t_0, in the TD methods, since the expected value of P_{t_1} at time t_0 is the forward start contract, the best estimate for the prediction function P_{t_0} is the value of the forward start contract $C^F(t_0, t_1; T)$ given in Equation (16.3.29). Thus, by analogy to finance, the goal of an agent in TD learning is to compute, at each time t, the forward start contract of a payoff starting at $t+1$ with maturity T, and to roll his position as he moves in time. Relating this to the predictive state representation theory in Section (16.3.1), the forward start contract seen at time t of a payoff starting at $t+1$ with maturity T is the question network, and the expected value seen at time $t+1$ of a payoff at maturity T is the answer network. Put another way, at every time t, the agent is learning the forward density function for the process X_t

$$\phi(x_{t_1}; T - t_1)$$

as given in Definition (16.3.1). Thus, as we move along a trajectory, we are learning the joint-distribution of the process via a question and answer relation. While in a parametric model the forward density function is given by the stochastic differential equation for the process X_t, in the TD-model (more generally in PSRs), it is inferred from the data.

16.4 Deep BSDE methods

Numerous financial problems such as option pricing can be expressed in terms of backward stochastic differential equation (BSDE) (see Pardoux et al. [1990]) (see Appendix (28.4)). For instance, in a complete market, the dynamics of the value of the replicating portfolio are given by a BSDE with linear generator f, with Z corresponding to the hedging portfolio. In the case of incomplete markets, one can not always build a replicating portfolio reaching the final wealth, and the price can not be determined by no-arbitrage arguments. In that case, the replication error is the tracking error. El Karoui et al. [1995] considered a stochastic control approach to the pricing of contingent claims in incomplete markets. They introduced the superstrategy (strategies with a positive tracking error) (see Appendix (28.4.3)) and defined the upper price for each contingent claim H as the smallest investment superhedging the claim H. They showed that this upper price was equal to the value function of a control problem. Following this approach, forward-backward stochastic differential equations (FBSDE) developed for solving high-dimensional parabolic partial differential equations (PDEs) (see Pardoux et al. [1999]). Recently, Han et al. [2017] [2018] introduced forward deep learning, called deep BSDE methods for solving uncoupled FBSDE. They considered approximating the gradient vector of the BSDE at each time step with a multilayer feedforward neural network. Alternatively, Raissi [2018] approximated the unknown solution to high-dimensional PDEs via coupled FBSDE with a deep neural network. The gradient vector was obtained by applying the chain rule for differentiating compositions of functions using automatic differentiation. Hientzsch [2019] provided an overview on the types of problems that could be solved with Deep BSDE. Ganesan et al. [2020] considered pricing barrier options with Deep BSDE by adding two types of nodes to the graph: one type to explicitly monitor the barrier conditions for each trajectory, the other type to preserve the time, state variables, and trading strategy at barrier breach or at maturity.
We are going to briefly introduce the FBSDE before presenting Deep BSDE. We will then relate the forward-backward stochastic neural networks introduced by Raissi [2018] to the TDBP-model described in Section (16.2.3).

16.4.1 Preliminary

16.4.1.1 Financial contracts

A derivative, or contingent claim, is a financial instrument whose value depends on the values of other more basic underlying variables being the prices of traded securities (see Hull [2012]). One of the simplest and most commonly traded instrument in the market is the forward contract. A forward contract is an agreement to buy or sell an asset at a certain future time for a certain price called the delivery price. At the time the contract is entered into, the delivery

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

price is such that the value of the forward contract is zero to both parties, so that neither party pays any money to obtain the contract. Given a continuous stochastic process $\{X_t\}_{t\geqslant 0}$, the payoff at time T, denoted $g(X_T)$, from a long position in a forward contract on one unit of an asset is

$$g(X_T) = X_T - K$$

where K is the delivery price and X_T is the spot price of the asset at maturity of the contract.
European, or vanilla, options such as a call option give the holder the right to buy the underlying asset by a certain date called the maturity for a specified price called the exercise price or strike price, while the put option gives him the right to sell it. We let K be the strike price and X_T be the underlying price at time T, then the payoff from a long position in a European call option is

$$g(X_T) = \max(X_T - K, 0) = (X_T - K)^+ \tag{16.4.32}$$

so that the option will be exercised if $X_T > K$.

16.4.2 European options in the constrained case

El Karoui et al. [1995] showed that the upper price of a superstrategy is not a solution of a BSDE (see Appendix (28.4.3)), but can be expressed as an increasing limit of penalised price processes which are solutions of non-linear BSDEs. They further showed that the duality between the hedging problem and the pricing one corresponds to a general duality between convex BSDEs and some control problems. This is illustrated with the pricing of European options in the constrained case.
In a general setting, the wealth equation is given by the BSDE

$$-dX_t = b(t, X_t, \sigma_t^\top \pi_t)dt - \pi_t^\top \sigma_t d\widehat{W}_t \tag{16.4.33}$$

where b is a real process defined on $\Omega \times [0,T] \times \mathbb{R} \times \mathbb{R}^n$ satisfying the standard assumption of a generator, and π_t is the portfolio process (vector describing the amount to be invested in each stock). In classical finance, the process b is given by the linear functional (see Theorem (28.4.3))

$$b(t, x, z) = -r_t x - z^\top \lambda_t$$

where λ is the bounded risk premium vector and r is the spot rate process (see Equation (28.2.12)). Given an initial investment x and a risky portfolio π, there exists a unique wealth process solution to the above forward equation with initial value x. In that setting, Z in Equation (28.4.24) is given by

$$Z_t = \sigma_t^\top \pi_t$$

In the special case where the underlying asset is driven by the BS-model, the portfolio of risky assets π_t is given by Equation (28.2.14).
When the terminal value is H, we get

$$X_t = H + \int_t^T b(s, Y_s, Z_s)ds - \int_t^T Z_s^\top d\widehat{W}_s$$

We let the price system be the map, Π, mapping the contingent claim $H \geqslant 0$ onto its price process $(\Pi_t(H);\ 0 \leqslant t \leqslant T)$ which is positive at any time t, is an increasing function with respect to H, and satisfies the no-arbitrage condition. That is, if $H_1 \geqslant H_2$, and if the prices $X_{1,t}$ and $X_{2,t}$ coincide on an event $\mathcal{A} \in \mathcal{F}_t$, then on $\mathcal{A}$, we get $H_1 = H_2$, a.s. Further, the price system Π is admissible for an agent (seller) if at any time t, the price $\Pi_t(H)$ is a convex function with respect to H.
Given $H \geqslant 0$ a square-integrable contingent claim, its price is the wealth process X associated with an admissible strategy financing H. That is, $(X, \sigma^\top \pi)$ is the square-integrable solution of the BSDE in Equation (16.4.33) with

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

standard parameters (b, H). Under some conditions, the price rule $\Pi(H) = X$ define an admissible price system for the seller. From the comparison theorem the following properties hold: the price is increasing with respect to H and the no-arbitrage conditions hold. Further, if b is convex with respect to (x, z), then the price system is convex with respect to the terminal value. This property holds when the prices are superprices.
When b is convex with respect to (x, z), we can give a variational formulation of the price system. We let $B(t, \beta, \gamma)$ be the polar process associated to b:

$$B(t, \beta, \gamma) = \inf_{(x,z)\in\mathbb{R}\times\mathbb{R}^n} \{b(t, x, z) + \beta x + \gamma^\top z\}$$

The effective domain of B is given by $\mathcal{D}_{t,B} = \{(\beta, \gamma)|\ B(t, \beta, \gamma) > -\infty\}$. Since b is convex, (b, B) satisfies the conjugation relation

$$b(t, x, z) = \sup\{B(t, \beta, \gamma) - \beta x - \gamma z|\ (\beta, \gamma) \in \mathcal{D}_{t,B}\}$$

The results on concave BSDE give the following proposition:

***Proposition* 11** *Let (X, π) be the hedging strategy for H such that $(X, \sigma^\top\pi)$ is the unique solution of the BSDE in Equation (16.4.33) with convex standard parameters (b, H). Then, X_t can be written as the maximum of ex post prices over all feasible deflators. That is,*

$$X_t = \operatorname{ess\,sup}\{X_{t,\beta,\gamma}|\ (\beta, \gamma) \in \mathcal{A}\}$$

where $\mathcal{A}$ is the set of feasible control parameters (β, γ) defined by

$$\mathcal{A} = \{(\beta, \gamma) \in \mathcal{P}|\ E[\int_0^T B^2(t, \beta_t, \gamma_t)dt] < \infty\}$$

and where, for each pair of control parameters $(\beta, \gamma) \in \mathcal{A}$, the ex post strategy $(X_{\beta,\gamma}, \pi_{\beta,\gamma})$ corresponds to the unique solution of the LBSDE

$$\begin{aligned} -dX_{t,\beta,\gamma} &= \big(B(t, \beta_t, \gamma_t) - \beta_t X_{t,\beta,\gamma} - (\gamma_t)^\top\sigma_t^\top\pi_{t,\beta,\gamma}\big)dt - \pi_{t,\beta,\gamma}^\top\sigma_t d\widehat{W}_t \\ X_{T,\beta,\gamma} &= H \end{aligned}$$

The ex post strategy $(X_{\beta,\gamma}, \pi_{\beta,\gamma})$ is a classical hedging strategy against the claim H in a fictitious market, with bounded interest rate process β, bounded risk premium process γ, and cost function B. Further, the price of the claim H is the standard price in an optimal fictitious market associated with $(\overline{\beta}, \overline{\gamma})$ which reaches the supremum in the conjugacy relation

$$b(t, X_t, \sigma_t^\top\pi_t) = B(t, \overline{\beta}, \overline{\gamma}) - \overline{\beta}_t X_t - (\overline{\gamma}_t)^\top\sigma_t^\top\pi_t\ ,\ d\mathbb{P}\otimes dt \text{ a.s.}$$

***Remark* 16.4.1** *The only difference with the non-constrained case is the fact that the optimal fictitious market depends on the claim to be priced and the introduction of a cost function.*

One can formulate in the same way the problems with constraints on the wealth or on the portfolio, but with a generator which can be infinite, with non-bounded effective domain. For instance, in the case of a portfolio process π_t constrained to take values in a convex set K, we get an upper price $(X_t;\ 0 \leqslant t \leqslant T)$ solution of the above BSDE with generator

$$b(t, x, \sigma_t^\top\pi) = -r_t x - \pi^\top\sigma_t\lambda_t + I_{\{K\}}(\pi)$$

where $I_{\{K\}}(\pi)$ is the indicator function of K (equal to 0 if $\pi \in K$ and equal to ∞ otherwise). The example of an incomplete market corresponds to $K = \{\theta \in \mathbb{R}^n|\ \pi_k = 0\ ,\ j \leqslant k \leqslant n\}$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

16.4.3 Introducing the FBSDE

Pardoux et al. [1990] [1999] have related parabolic PDEs to the BSDEs (see Appnedix (28.4)), so that one can solve the latter rather than the former. For instance, Markovian BSDEs give a nonlinear Feynman-Kac representation of some nonlinear parabolic PDEs. Han et al. [2018] proposed to solve the forward BSDE (FBSDE) by approximating the spatial gradients with neural networks.
Before describing the Deep BSDE, we first present the forward BSDE framework. We consider the semilinear parabolic PDE

$$\frac{\partial u}{\partial t}(t,x) + Lu(t,x) + f\big(t,x,u(t,x),a^\top(t,x)\nabla_x u(t,x)\big) = 0$$

with

$$Lu(t,x) = \frac{1}{2}Tr\big(a(t,x)a^\top(t,x)\nabla_x^2 u(t,x)\big) + \nabla_x u(t,x)b(t,x)$$

defined on a domain $D \subset \mathbb{R}^d$ and $x \in D$. Note, $Du(t,x) = \nabla_x u(t,x)$ is the gradient vector at the point x, and $D^2u(t,x) = \nabla_x^2 u(t,x)$ is the Hessian matrix at the point x. We consider the terminal condition

$$u(T,x) = g(T,x)$$

for a given function g. We are interested in the solution at $t = 0$ and single space-point $x = \xi$ for some vector $\xi \in \mathbb{R}^d$. We consider the Ito process $\{X; t \in [0,T]\}$ with dynamics given by

$$dX_t = b(t,X_t)dt + a(t,X_t)dW_t$$

where $X_t \in \mathbb{R}^d$, $dW_t \in \mathbb{R}^d$, $b(t,x) \in \mathbb{R}^d$, and $a(t,x) \in \mathbb{R}^{d\times d}$, with uncorrelated Brownian motions W. Note, the correlation is expressed via the diffusion term $a(t,x)$. We let $X^{t,x}$ be a version of X starting at time t in x. It follows directly from Ito's formula that $Y_t = u(t,X_t)$ satisfies the BSDE (see Cheridito et al. [2007])

$$dY_t = -f\big(t,X_t,Y_t,a^\top(t,X_t)\nabla_x u(t,X_t)\big)dt + \big(\nabla_x u(t,X_t)\big)^\top a(t,X_t)dW_t$$

The above terminal condition for the PDE becomes $Y_T = g(T,X_T)$ for the BSDE. If we let $\theta_t = \nabla_x u(t,X_t)$ be the strategy, we can rewrite the BSDE as

$$dY_t = -f\big(t,X_t,Y_t,a^\top(t,X_t)\theta_t\big)dt - \theta_t^\top a(t,X_t)dW_t$$

This BSDE together with the dynamics of X describe the dynamics of the solution $u(t,x)$ to the above PDE. It is called the forward BSDE (FBSDE) (see Ma et al. [1994]). So, rather than solving the PDE, we can find the stochastic processes Y_t and θ_t. Given a partition of the time interval $[0,T] : 0 = t_0 < t_1 < \cdots < t_N = T$, we apply the Euler-Maruyama discretisation for X_t and Y_t, getting

$$\begin{aligned} X_{t_{i+1}} &= X_{t_i} + b(t_i,X_{t_i})\Delta t_i + a(t_i,X_{t_i})\Delta W^i \\ Y_{t_{i+1}} &= Y_{t_i} - f\big(t_i,X_{t_i},Y_{t_i},a^\top(t_i,X_{t_i})\theta_{t_i}\big)\Delta t_i + \pi_{t_i}^\top a(t_i,X_{t_i})\Delta W^i \end{aligned} \tag{16.4.34}$$

where $\Delta t_i = t_{i+1} - t_i$ and $\Delta W^i = W_{t_{i+1}} - W_{t_i}$, for $i = 1,...,N-1$. We can then use this discretisation to simulate forward the processes $\{X_{t_i}; 0 \leqslant i \leqslant N\}$ and $\{Y_t; 0 \leqslant i \leqslant N\}$.
Note, the above BSDE corresponds to the BSDE in Equation (28.4.24) with $Z_t = a^\top(t,X_t)\theta_t$. In the special case where the process X follows the dynamics of a local volatility model

$$dX = r(t,x)Xdt + \sigma(t,x)XdW$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $r(t,x)$ is the risk-free rate and $\sigma(t,x)$ is the local volatility, we can apply Ito's formula to the function $y(t,x)$ we get

$$dy(t,x) = r(t,x)y(t,x)dt + \sigma(t,x)x\frac{\partial y}{\partial x}dW$$

We can then identify terms to the BSDE as follows: $Y_t = y(t,X_t)$, $\theta_t = \frac{\partial y}{\partial x}(t,X_t)$, $a(t,X_t) = \sigma(t,X_t)X_t$, and $f(t,X_t,Y_t,Z_t) = -r(t,X_t)Y_t$.
More generally, we can set $\pi_t = \theta_t X_t$ and write the BSDE as

$$dY_t = -f\big(t,Y_t,\sigma_t^\top \pi_t\big)dt - \pi_t^\top \sigma_t dW_t$$

with

$$dX_t = b(t,X_t)dt + \sigma_t X_t dW_t$$

where σ_t is a stochastic process at time t.

16.4.4 Deep BSDE

16.4.4.1 Description

In order to solve the above FBSDE, Han et al. [2018] proposed to discretise the SDEs (see Equation (16.4.34)) and represent the process Z_t with the functions $p_i(X_{t_i})$, one for each time t_i. Then, each function $p_i(X_{t_i})$ is given a deep neural network. That is, rather than finding the process Z_t, the authors considered approximating the function $x \to a^\top(t,X_t)\theta_t$ at each time step $t = t_i$ with a multilayer feedforward neural network

$$a^\top(t_i,X_{t_i})\theta_{t_i} \approx a^\top\theta(t_i,X_{t_i}\,|\psi_i)$$

where ψ_i denotes the parameters of the neural network at time $t = t_i$. All the subnetworks for $i = 1,...,N-1$ are stacked together to form a deed neural network as a whole. This network takes the paths $\{X_{t_i};0 \leqslant i \leqslant N\}$ and $\{W_{t_i};0 \leqslant i \leqslant N\}$ as input data and gives the final output, denoted by $\widehat{u}\big(\{X_{t_i};0 \leqslant i \leqslant N\},\{W_{t_i};0 \leqslant i \leqslant N\}\big)$, as an approximation to the value $u(t_N,X_{t_N})$. Using the terminal condition, they defined the expected loss function as the L_1 norm

$$l(\psi) = E\big[\big|g(t_N,X_{t_N}) - \widehat{u}\big(\{X_{t_i};0 \leqslant i \leqslant N\},\{W_{t_i};0 \leqslant i \leqslant N\}\big)\big|^2\big]$$

In order to numerically compute $Y_0 = u(0,X_0)$, the authors treated $u(0,X_0) \approx \psi_{u_0}$ and $\theta_0 = \nabla_x u(0,X_0) \approx \psi_{\nabla u_0}$ as parameters in the model, and used the BSDE as a way of computing u at maturity T, knowing $u(0,X_0)$ and $\nabla_x u(t,X_t)$. In that setting, the total set of parameters is $\psi = \{\psi_{u_0},\psi_{\nabla u_0},\psi_1,...,\psi_{N-1}\}$. It is optimised with stochastic optimisation methods such as mini-batch stochastic gradient descent combined with Adam optimisers to determine the parameters of networks p_n and Y_0.
For simplicity of exposition, they represented the strategy θ_t with the functions $p_n(X_{t_n}) = \theta(t_n,X_{t_n})$, one for each time t_n, for $n = 1,...,N-1$. Then, each function $p_n(X_{t_n})$ is given a deep neural network. We first need to generate a realisation of X_{t_n}, and given Y_0 and all the $\theta_{t_n} = p_n$ networks, we compute iteratively Y_{t_n} to reach $Y_T = Y_{T_N}$. We let H be the number of hidden layers, and $h_n^1,...,h_n^H$ be the intermediate neurons in the subnetwork at time t_n. Each column for $t_1,t_2,...,t_{N-1}$ corresponds to a subnetwork. We display the architecture of the whole network in Figure (16.11).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 16.11: Network architecture for the Deep BSDE.

There are three types of connections in the network:

1. $X_{t_n} \to h_n^1 \to \cdots \to h_n^H \to \nabla u(t_n, X_{t_n})$ is a multilayer feedforward neural network approximating the spatial gradients at time $t = t_n$. We want to optimise the weights ψ_n of this subnetwork.
2. $\big(u(t_n, X_{t_n}), \nabla u(t_n, X_{t_n}), W_{t_{n+1}} - W_{t_n}\big)$ is the forward iteration giving the final output of the network as an approximation to $u(t_N, X_{t_N})$.
3. $\big(X_{t_n}, W_{t_{n+1}} - W_{t_n}\big) \to X_{t_{n+1}}$ is the shortcut connecting blocks at different times.

The whole network has $(H+1)(N-1)$ layers with free parameters to be simultaneously optimised. If we want to compute the solution u in a region $D \subset \mathbb{R}^d$ at time $t = 0$, the authors suggested to let X_0 be a D-valued random variable and use two additional networks with parameters $\{\psi_{u_0}, \psi_{\nabla u_0}\}$ to approximate the functions $x \to u(0, x)$ and $x \to \nabla_x u(0, x)$.
The authors implemented this network with each subnetwork having four layers, with one input layer (d dimensional), two hidden layers (both $d + 10$ dimensional), and one output layer (d dimensional). The activation was the ReLu and a technique of batch normalisation 30 was used in the subnetworks just after each linear transformation and before activation. They used batches of size 64 and they computed the mean and the standard deviation of the relative L_1 approximation errors with five independent runs of the algorithm with different random seeds. They needed about $4,000$ iteration steps to compute a European claim in the BS-framework based on 100 underlying assets conditional on no default.

Similarly, Ganesan et al. [2020] proposed to discretise the SDEs (see Equation (16.4.34)) and represent the strategy θ_t with the functions $p_n(X_{t_n}) = \theta(t_n, X_{t_n})$, one for each time t_n. Instead of using a constant X_0, the system is initialised with random values. Here, $Y_0(X_0)$ is replaced with a network at time t_0 with parameters ψ_0. The authors used the L_2 norm of the replication error

$$Y_{T_N} - g(t_N, X_{t_N})$$

as a loss function. It is optimised with stochastic optimisation methods such as mini-batch stochastic gradient descent combined with Adam optimisers to determine the parameters of networks p_n and Y_0. We display in Figure (16.12) the graph of the forward method for European options with random X_0.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 16.12: Network architecture for the Deep BSDE with random X_0.

Since θ_t is obtained during the optimisation process described above, it is an approximation to the time-continuous case $\theta_t = \nabla_x u(t, X_t)$.

16.4.4.2 Drawbacks

Note, parametric models for computing conditional expectations of future events are tied to their ability of capturing the dynamics of the underlying stochastic processes. As a result, their misspecification will lead to computation errors, since their formulae depend on the particular form of the dynamics of the stochastic processes chosen. On the other hand, non-parametric models, such as neural networks (NNs), use data to estimate the implicit stochastic processes driving the dynamics of the future events and their relationship with conditional expectations. The Deep BSDE methods are semiparametric models, and as such, still suffer from having to specify the dynamics of the underlying stochastic processes.

The number of parameters to estimate grows with number of points N used to discretise time, which becomes prohibitive for large maturity T, or when Δt needs be very small to improve convergence of the Euler-Maruyama scheme. Further, since the method only provide solution at $t = 0$ and single space-point $x = \xi$, the algorithm will need to be retrain to compute $Y_t = u(t, X_t)$ at a later time $t > 0$. Finally, one should not let Y_0 and Z_0 act as parameters in the model.

16.4.5 Forward-backward stochastic neural networks

16.4.5.1 Description

Raissi [2018] considered the coupled FBSD of the form

$$\begin{aligned} dX_t &= \mu(t, X_t, Y_t, Z_t)dt + \sigma(t, X_t, Y_t)dW_t \text{ , } t \in [0, T] \\ X_0 &= \xi \\ dY_t &= f(t, X_t, Y_t, Z_t)dt + Z_t^\top \sigma(t, X_t, Y_t)dW_t \text{ , } t \in [0, T] \\ Y_T &= g(X_T) \end{aligned}$$

where W_t is a vector of Brownian motions. A solution to these SDEs are the stochastic processes (X_t, Y_t, Z_t). These FBSDEs are related to quasi-linear PDEs of the form

$$u_t = L(t, x, u, Du, D^2u)$$

with terminal condition $u(T, x) = g(x)$ and

$$L(t, x, y, z, \gamma) = f(t, x, y, z) - \mu^\top(t, x, y, z)z - \frac{1}{2}TR\big[\sigma\sigma^\top(t, x, y)\gamma\big]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where Du is the gradient vector at the point x, and D^2u is the Hessian matrix at the point x. From Ito's formula we get the relation (see Cheridito et al. [2007])

$$Y_t = u(t, X_t) \text{ and } Z_t = Du(t, X_t)$$

Combining classical numerical solutions to PDEs, which expand the unknown solution $u(t, x)$ with a set of basis functions, and machine learning, which uses neural networks, the author directly approximated the unknown solution $u(t, x)$ with a neural network. Once trained to represent $u(t, x)$, the network can be evaluated anywhere in the time-space domain. The gradients $Du(t, x)$ are computed by differentiating the network using automatic differentiation (see Baydin et al. [2015]).
Given a partition of the time interval $[0, T] : 0 = t_0 < t_1 < \cdots < t_N = T$, the Euler-Maruyama scheme is applied to the above set of FBSDEs, getting

$$\begin{aligned} X_{n+1} &= X_n + \mu(t_n, X_n, Y_n, Z_n)\Delta t_n + \sigma(t_n, X_n, Y_n)\Delta W_n \\ Y_{n+1} &= Y_n + f(t_n, X_n, Y_n, Z_n)\Delta t_n + (Z_n)^\top \sigma(t_n, X_n, Y_n)\Delta W_n \end{aligned}$$

The loss function is given by

$$l(\psi) = \sum_{m=1}^{M} \sum_{n=0}^{N-1} A_n^m + \sum_{m=1}^{M} B_N^m$$

where

$$A_n^m = \left|Y_{n+1}^m - Y_n^m - F_n^m \Delta t_n - (Z_n^m)^\top \Sigma_n^m \Delta W_n^m\right|^2 \text{ and } B_N^m = \left|Y_N^m - g(X_N^m)\right|^2$$

for M different realisations of the underlying Brownian motions. Here, $F_n^m = f(t_n, X_n^m, Y_n^m, Z_n^m)$ and $\Sigma_n^m = \sigma(t_n, X_n^m, Y_n^m)$, where m corresponds to the mth realisation of the trajectory. Since $Y_n^m = u(t_n, X_n^m)$ and $Z_n^m = Du(t_n, X_n^m)$, the loss function depends on the vector of parameters ψ of the neural network approximating $u(t, x)$. Finally, we have

$$X_{n+1}^m = X_n^m + \mu(t_n, X_n^m, Y_n^m, Z_n^m)\Delta t_n + \Sigma_n^m \Delta W_n^m$$

with $X_0^m = \xi$ for every m.

16.4.5.2 Link to the TD-procedures

In the above loss function, the term B_N^m is the L_1 norm on the matching error at maturity T used in the Deep BSDE in Section (16.4.4), and the term A_n^m corresponds to the agent predicting the value Y_{n+1}^m at time t_n, given the mth trajectory. Here, his best estimate, $\widehat{Y}_{n+1}^m$, along the mth trajectory is

$$\widehat{Y}_{n+1}^m = Y_n^m + F_n^m \Delta t_n + (Z_n^m)^\top \Sigma_n^m \Delta W_n^m$$

which is given by the dynamics of Y_t. This is to be related to the TD-procedures described in Section (16.1.3). Since $Y_t = u(t, X_t)$ is the solution to the above PDE, at time t and for the point X_t, its value is a conditional expectation with terminal condition $g(X_T)$. Hence, at each time t_n, the agent is learning a conditional expectation. As a result, this loss function is similar to the loss function for the TD-procedures given in Equation (16.1.14). Finally, since the forward-backward stochastic neural networks is approximated with a neural network, it is to be related the TDBP-model described in Section (16.2.3). At each time t_n, along the mth trajectory, we have the pseudo TD-error given by

$$\sigma_n^m = Y_{n+1}^m - \widehat{Y}_{n+1}^m$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

which is the error term in value-learning.
As discussed in Section (16.3.2), in the TD-procedures, at each time step t_n, the agent is learning to predict the forward start contract given in Equation (16.3.29) in a model independent way. However, in the forward-backward stochastic neural networks, the agent relies on a dynamical system to predict the forward start contract at time t_n. In the former the agent is learning the forward density function for the process X_t from the data, while in the latter, the agent is a priori given that forward density from the dynamics of the process X_t.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Annexes

16.5 Classical conditioning

See book by Chance [2008].

16.5.1 Overview

Classical conditioning (also known as Pavlovian or respondent conditioning) refers to a learning procedure in which a biologically potent stimulus (such as food) is paired with a previously neutral stimulus (such as a bell). Classical conditioning occurs when a conditioned stimulus (CS) is paired with an unconditioned stimulus (US). Usually, the conditioned stimulus is a neutral stimulus (such as the sound of a tuning fork), the unconditioned stimulus is biologically potent (such the taste of food) and the unconditioned response (UR) to the unconditioned stimulus is an unlearned reflex response (such salivation). After pairing is repeated the organism exhibits a conditioned response (CR) to the conditioned stimulus when the conditioned stimulus is presented alone. (A conditioned response may occur after only one pairing.) Thus, unlike the UR, the CR is acquired through experience, and it is also less permanent than the UR.
Classical conditioning differs from operant or instrumental conditioning: in classical conditioning, behaviours are modified through the association of stimuli as described above, whereas in operant conditioning behaviours are modified by the effect they produce (such as reward or punishment).
In Pavlov's experiments, the unconditioned stimulus (US) was the dog food because its effects did not depend on previous experience. The metronome's sound is originally a neutral stimulus (NS) since it does not elicit salivation in the dogs. After conditioning, the metronome's sound becomes the conditioned stimulus (CS) or conditional stimulus; since its effects depend on its association with food. Similarly, the responses of the dog follow the same conditioned-versus-unconditioned arrangement. The conditioned response (CR) is the response to the conditioned stimulus, whereas the unconditioned response (UR) corresponds to the unconditioned stimulus.

16.5.2 Some conditioning

16.5.2.1 Forward conditioning

Learning is fastest in forward conditioning. During forward conditioning, the onset of the CS precedes the onset of the US in order to signal that the US will follow. Two common forms of forward conditioning are delay and trace conditioning:

- Delay conditioning: The CS is presented and is overlapped by the presentation of the US. For example, if a person hears a buzzer for five seconds, during which time air is puffed into their eye, the person will blink. After several pairings of the buzzer and the puff, the person will blink at the sound of the buzzer alone. This is delay conditioning.
- Trace conditioning: During trace conditioning, the CS and US do not overlap. Instead, the CS begins and ends before the US is presented. The stimulus-free period is called the trace interval, or, the conditioning interval. If in the above buzzer example, the puff came a second after the sound of the buzzer stopped, that would be trace conditioning, with a trace or conditioning interval of one second.

16.5.2.2 Second-order or higher-order conditioning

Second-order or higher-order conditioning follow a two-step procedure. First a neutral stimulus (CS1) comes to signal a US through forward conditioning. Then a second neutral stimulus (CS2) is paired with the first (CS1) and comes to yield its own conditioned response. For example: A bell might be paired with food until the bell elicits salivation. If a light is then paired with the bell, then the light may come to elicit salivation as well. The bell is the CS1 and the food is the US. The light becomes the CS2 once it is paired with the CS1.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

16.5.2.3 Backward conditioning

Backward conditioning occurs when a CS immediately follows a US. Unlike the usual conditioning procedure, in which the CS precedes the US, the conditioned response given to the CS tends to be inhibitory. This presumably happens because the CS serves as a signal that the US has ended, rather than as a signal that the US is about to appear. For example, a puff of air directed at a person's eye could be followed by the sound of a buzzer.

16.5.2.4 Temporal conditioning

In temporal conditioning, a US is presented at regular intervals, for instance every 10 minutes. Conditioning is said to have occurred when the CR tends to occur shortly before each US. This suggests that animals have a biological clock that can serve as a CS. This method has also been used to study timing ability in animals (see Animal cognition).
For example, a US such as food to a hungry mouse is simply delivered on a regular time schedule such as every thirty seconds. After sufficient exposure the mouse will begin to salivate just before the food delivery. This then makes it temporal conditioning as it would appear that the mouse is conditioned to the passage of time. (see Basic Principle of Learning).

16.5.2.5 Extinction procedure

In the extinction procedure, the CS is presented repeatedly in the absence of a US. This is done after a CS has been conditioned by one of the methods above. When this is done, the CR frequency eventually returns to pre-training levels. However, extinction does not completely eliminate the effects of the prior conditioning. This is demonstrated by spontaneous recovery, when there is a sudden appearance of the (CR) after extinction occurs, and other related phenomena. These phenomena can be explained by postulating accumulation of inhibition when a weak stimulus is presented.

16.5.3 The Rescorla-Wagner model

The Rescorla-Wagner (RW) model (see Rescorla [1967]) is a relatively simple yet powerful model of conditioning. The model predicts a number of important phenomena, but it also fails in important ways, thus leading to a number of modifications and alternative models. However, because much of the theoretical research on conditioning in the past 40 years has been instigated by this model or reactions to it, the RW model deserves a brief description here.
The Rescorla-Wagner model argues that there is a limit to the amount of conditioning that can occur in the pairing of two stimuli. One determinant of this limit is the nature of the US. For example, pairing a bell with a juicy steak is more likely to produce salivation than pairing the bell with a piece of dry bread, and dry bread is likely to work better than a piece of cardboard. A key idea behind the RW model is that a CS signals or predicts the US. One might say that before conditioning, the subject is surprised by the US. However, after conditioning, the subject is no longer surprised, because the CS predicts the coming of the US. Note that the model can be described mathematically and that words like predict, surprise, and expect are only used to help explain the model. Here the workings of the model are illustrated with brief accounts of acquisition, extinction, and blocking. The model also predicts a number of other phenomena.
The equation is

$$\Delta V = \alpha\beta\big(\lambda - \Sigma V\big)$$

where V represents the current associative strength of the CS, and ΔV is the change in this strength that happens on a given trial. ΣV is the sum of the strengths of all stimuli present in the situation, λ is the maximum associative strength that a given US will support; its value is usually set to 1 on trials when the US is present, and 0 when the US is absent. α and β constants related to the salience of the CS and the speed of learning for a given US. This equation specifies the amount of learning that will occur on a single pairing of a conditioning stimulus (CS) with an unconditioned stimulus (US). It is solved repeatedly to predict the course of learning over many such trials.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 17

Reinforcement learning

17.1 Learning the optimal policy

17.1.1 An overview

We have defined in Section (15.3) a Markov decision process (MDP) as a 4-tuple $(\mathcal{S}, \mathcal{R}, \mathbb{P}, \gamma)$ comprising a function $R(s, a)$ that returns the reward received for taking action a in state s, a transition probability function $P(s^{'}|s, a)$, specifying the probability that the environment will transition to state $s^{'}$ if the agent takes action a in state s, and the probability $\pi(s, a)$ of taking certain actions from certain states. As such the MDP is a probabilistic model.
The goal of reinforcement learning (LR) is to find a policy π that maximises the expected future (discounted) reward of an agent. To do so we can learn

1. the optimal policy π^* of that agent, or, alternatively,
2. the optimal value function $V^*(s)$.

This is because we can always derive π^* from $V^*(s)$ and vice versa, since the optimal policy satisfies $s \rightarrow a^*(s)$ where the optimal action $a^*(s)$ is defined in Equation (15.3.17). That is, learning optimal actions for all states simultaneously means learning a policy, which is our objective.

17.1.1.1 The RL problem

Knowing all elements of a Markov decision process (MDP), we can just compute the solution before ever actually executing an action in the environment. In machine learning (ML), we typically call "planning" the act of computing the solution to a decision-making problem before executing an actual decision. In general there is no closed form solution to the associated Bellman equation, and one must rely on some iterative solutions. As discussed in Section (15.3.3.2), some classic planning algorithms for MDPs include Value Iteration, Policy Iteration, among others.

What makes a problem an RL problem, rather than a planning problem, is that the agent does not know how the world will change in response to its actions (the transition function P), nor what immediate reward it will receive for doing so (the reward function R). The agent will simply have to try taking actions in the environment, observe what happens, and somehow, find a good policy from doing so. Some of the solutions available to the agent are as follows:

- Model-based RL algorithms: The agent can learn a model of how the environment works from its observations and then plan a solution using that model. That is, if the agent is currently in state s_1, takes action a_1, and then observes the environment transition to state s_2 with reward r_2. That information can be used to improve its estimate of $P(s_2|s_1, a_1)$ and $R(s_1, a_1)$, which can be performed using supervised learning approaches (see

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter (10)). Once the agent has adequately modelled the environment, it can use a planning algorithm with its learned model to find a policy. That is, replace the transition probability P and return R with the learned network F in the planning algorithm. Thus, when explicit models of transition probabilities and reward functions are learned, the RL solutions are model-based RL algorithms.

- Model-free RL algorithms: As it turns out though, the agent does not have to learn a model of the environment to find a good policy. One of the most classic examples is Q-learning, which directly estimates the optimal Q-values of each action in each state (roughly, the utility of each action in each state), from which a policy may be derived by choosing the action with the highest Q-value in the current state. Actor-critic and policy search methods directly search over policy space to find policies that result in better reward from the environment. Because these approaches do not learn a model of the environment they are called model-free algorithms.

17.1.1.2 Types of RL algorithms

17.1.1.2.1 Model-based versus model-free Put another way, the terms "model-based" and "model-free" strictly refer as to whether, whilst during learning or acting, the agent uses predictions of the environment response. That is, part or full probability distribution of the next states and next rewards. These predictions can be provided entirely outside of the learning agent, or they can be learned by the agent, in which case they will be approximate. For example

$$R^{'}(s_t, a_t) = R(s_t, a_t) + \alpha_t\big(r_{t+1} - R(s_t, a_t)\big)$$

Just because there is a model of the environment implemented, does not mean that a RL agent is model-based. To qualify as model-based, the learning algorithms have to explicitly reference the model. Thus, a simple way of checking whether an RL algorithm is model-based or model-free is to find out, after learning, if the agent can make predictions about what the next state and reward will be before it takes any action. If it can, then it is a model-based RL algorithm, otherwise it is a model-free algorithm. This same idea may also apply to decision-making processes other than MDPs. For instance, given the value function in Equation (15.3.14), the full Value Iteration algorithm

$$V_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(s, a)\big[R(s, a) + \gamma \sum_{s^{'} \in \mathcal{S}} P(s^{'}|s, a)V_t(s^{'})\big]$$

where $t \in \mathbb{N}$ is the iteration number, uses the transition probability $P(\cdot|s, a)$. As a result, it is a model-based algorithm. On the other hand, we can estimate this equation with the approximation

$$V_{t+1}(s_t) = r_{t+1} + \gamma V_t(s_{t+1})$$

which does not use any probabilities defined by the MDP. Note, r_{t+1} is just the reward that is obtained at the next time step (after taking the action), but it is not necessarily known beforehand. So, this approximation is a model-free algorithm.

17.1.1.2.2 Some examples Some examples of model-based and model-free algorithms are as follows:

- Model-based algorithms can:

  1. just use the model to plan (1) backpropagation to optimise over actions in continuous spaces (2) Monte Carlo tree search in discrete action spaces.
  2. backpropagate gradients into the policy.
  3. use the model to learn a value function.

  The archetypical model-based algorithms are Dynamic Programming (case [3]) (Policy Iteration and Value Iteration) or planning algorithms (case [1]) such as MCTS. All of these methods use the model's predictions or distributions of next state and reward in order to calculate optimal actions. Specifically in Dynamic Programming, the model must provide state transition probabilities, and expected reward from any state-action pair.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Model-free algorithms: Algorithms that purely sample from experience such as Monte Carlo Control, SARSA, Q-learning, Actor-Critic are model-free RL algorithms. They rely on real samples from the environment and never use generated predictions of next state and next reward to alter behaviour (although they might sample from experience memory, which is close to being a model).

17.1.1.2.3 Off-policy versus on-policy The RL algorithms can also be classified as off-policy methods and on-policy methods.

1. Off-policy methods: The policy used to generate behaviour, called the behaviour policy, may be unrelated to the policy that is evaluated and improved, called the estimation policy. An off-policy learner learns the value of the optimal policy independently of the agent's actions (from different action set). Thus, it is capable of improving the policy without generating new samples from that policy. One advantage of this separation is that the estimation policy may be deterministic (such as greedy policy), while the behaviour policy can continue to sample all possible actions. Q-learning is an off-policy learner.

2. On-policy methods: They estimate the value of a policy while using it for control. They learn action values not for the optimal policy, but for a near-optimal policy that still explores. An on-policy learner learns the value of the policy being carried out by the agent (current policy) including the exploration steps. Thus, they are dependent on the policy used. Each time the policy is modified, even slightly, we need to generate new samples. On the other hand, the updates to the networks are made at each time step during the trial, without the need to store any state-action information. SARSA is an on-policy learner. One solution is to use two policies, one that is learned about (target policy) and that becomes the optimal policy, and one that is more exploratory and is used to generate behaviour (behaviour policy).

17.1.1.3 Policy optimisation

Standard function approximations approximate the value function and determine a policy from it. However, it is oriented towards finding deterministic policies, and suffers from discontinuous changes in the estimated value making it difficult to converge. An alternative solution is to let the policy be explicitly represented by its own function approximator, and update it according to the gradient of expected reward with respect to the policy parameters. This is called policy gradient method. There are other policy optimisation methods, some of which are introduced in the next section.

17.1.1.3.1 The algorithms We classify the algorithms for policy optimisation into three broad categories:

1. Policy iteration methods: they alternate between estimating the value function under the current policy and improving the policy (see Bertsekas [2005]). It is a two step process

   (a) Policy evaluation: starting from the last policy updated in policy improvement, the value function is estimated for every state (or the Q-value function is estimated for every state and action).

   (b) Policy improvement: the policy is improved by setting its greedy policy.

   These two steps are repeated until convergence. This is illustrated in Figure (15.6).

2. Policy gradient methods: they use an estimator of the gradient of the expected cost function obtained from sample trajectories (see Peters et al. [2008]).

3. Derivative-free stochastic optimisation methods: examples are the cross-entropy method (CEM) and covariance matrix adaptation (CMA), which treat the cost as a black box function to be optimised in terms of the policy parameters (see Fu et al. [2005]). These methods are very good for dynamic programming and continuous control problems.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.1.1.3.2 Comparison One way of comparing algorithms is to consider

- their sample efficiency: the number of samples required to get a good policy.
- their stability and ease of use.

In general, off policy such as model-based algorithms are more efficient because they use fewer samples. On policy and evolutionary algorithms are less efficient.
When looking at stability and convergence, we get

- Value function fitting: in the best cases it minimises the error of fit and in the worst case it does not optimise anything. Full observability is assumed.
- Model-based: it minimises the error of fit and will converge. Often assume continuity or smoothness.
- Policy gradient: the only algorithm performing gradient descent on the true objective. Episodic learning is assumed.

17.1.1.3.3 Implementation These methods implement a single-step update of a current value of the Bellman equation. Since the state space and action space are both discrete, we get a value for each combination of a state and action. As such the MDP is represented in a tabulated form as a two-dimensional matrix (tensor). For example, in the case of 16 states and 4 actions per state we can store $V_\pi(s)$ in a table where $\mathcal{T}$ is $16 \times 16 \times 4$ tensor. For a time-dependent problem, a time argument is added to the function so that the table is represented by a three-dimensional tensor for a discrete state-action space.

17.1.1.4 Generalisation

As discussed in Section (16.1), there are two popular temporal difference (TD)-based RL algorithms, namely the AHC-learning (Adaptive Heuristic Critic) due to Sutton [1984] and Q-learning due to Watkins [1989]. The former learns an evaluation function of states, while the latter learns an evaluation function of state-action pairs. We devise a recursive definition of the target evaluation function and then incrementally construct a function satisfying this definition. Both V-learning and Q-learning require learning an evaluation function, the V-function and Q-function, respectively, which are stored in a lookup table. This solution is only feasible when learning problems having a small state space, and does not apply to very large and continuous state spaces. One must therefore rely on function approximators to save memory and to generalise the solution.
A large number of function approximator have been proposed, including

- neural networks
- decision trees
- instance-based function approximation by nearest neighbours or by locally weighted regression
- CMAC (Cerebellar Model Articulation Computer) (see Albus [1981])

However, these methods were designed to be used in supervise learning paradigm, that is, for each training input the corresponding desired output is directly known to the learning system. This is not the case in RL paradigm. In general, the real utilities of states and actions are never directly known to the learning agent, but can only be estimated through TD methods. Nonetheless, neural networks are the most popular function approximator in RL.
As discussed in Section (16.2), various authors proposed to combine the error backpropagation algorithm with TD methods to solve hard RL problems. That is, both the V-function and Q-function can be represented by neural networks and learned by temporal difference and backpropagation. Anderson [1987] combined backpropgation with AHC-learning in solving a pole balancing problem. Combining the two methods, Tesauro [1992] devised a program that

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

learned to play backgammon better than supervised learning. Lin [1993] combined the two methods to learn complex evaluation functions with a large number of binary and real-valued inputs.
A policy net takes a state as input and to each action a value, called action merit, indicating the relative merit of performing that action. Thus, the agent's policy is to choose the action having the highest merit. However, to learn an optimal policy effectively, the agent should sometime take suboptimal actions to assess the relative merits of actions. One can use a stochastic action selector that favours actions having high merits.

17.1.2 The policy iteration

We are now going to describe policy iteration for planning problem and then consider the reinforcement learning (RL) problem for some model-based RL algorithms as well as some model-free RL algorithms.

17.1.2.1 Introduction

In dynamic programming (DP), we assume that we know the transition matrix of the environment $P(s^{'}|s,a)$ and the expected next step reward $R(s,a)$ for all the possible discrete state-action pair (s,a). Given Remark (15.3.5), we want to find a greedy policy π^* that always seeks an action maximising the Q-value function in the current state.
Given Equation (15.3.23), one solution is to let the Advantage Function A_π define how much an action a is better than the expected action according to the policy π (see details in Section (17.1.5)). It is given by

$$\begin{aligned} A_\pi(s,a) &= Q_\pi(s,a) - V_\pi(s) \\ &= Q_\pi(s,a) - E_{a\sim\pi(s,a)}[Q_\pi(s,a)] \\ &= r(s,a) + \gamma E_{s'\sim p(s'|(s,a))}[V_\pi(s^{'})] - V_\pi(s) \end{aligned} \tag{17.1.1}$$

The main idea behind policy iteration follows the Greedy Policy Improvement (see Propositions (5) and (6)). It is based on the two-step process presented in Section (17.1.1.3). It can be described with the Algorithm (34):

Algorithm 34 Policy Iteration Algorithm

1: Estimate $A_\pi(s,a)$ (or V_π or Q_π).
2: Set $\pi \leftarrow \pi^{'}$ (policy update).
3: Repeat step [1] until convergence.

where

$$\pi^{'}(a,s) = \begin{cases} 1 \text{ if } a = \arg\max_{a\in\mathcal{A}} A_\pi(s,a) \\ 0 \text{ otherwise} \end{cases}$$

Thus, we need to estimate the value function $V_\pi(s)$. Note, this algorithm is a kind of gradient ascent. Since it searches among the finite deterministic policies, it can be proved to converge after a finite number of steps. However, evaluating the value function at each step makes it very costly for large number of states.
We can also replace the optimisation of the Advantage Function with that of the Q-value function $Q_\pi(s,a)$, getting $\arg\max_{a\in\mathcal{A}} Q_\pi(s,a)$. In that case, we get the algorithm

Algorithm 35 Policy Q-Value Iteration Algorithm

1: Set $Q(s,a) \leftarrow r(s,a) + \gamma E[V_\pi(s^{'})]$.
2: Set $V_\pi(s) \leftarrow \max_a Q(s,a)$.
3: Repeat step [1] until convergence.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Some examples of policy are:

- Final policy
$$\pi(a,s) = \begin{cases} 1 \text{ if } a = \arg\max_a Q_\pi(s,a) \\ 0 \text{ otherwise} \end{cases}$$
- Epsilon-greedy
$$\pi(a,s) = \begin{cases} 1-\epsilon \text{ if } a = \arg\max_a Q_\pi(s,a) \\ \frac{\epsilon}{(|\mathcal{A}|-1)} \text{ otherwise} \end{cases}$$
- Boltzmann exploration
$$\pi(a,s) \propto e^{Q_\pi(s,a)}$$

Given the Bellman optimality equation (BOE) in Equation (15.3.15) and, using the matrix notation in Section (15.3.2), we can define an operator $\mathcal{B}$ as follows:

$$\mathcal{B}V = \max_{a\in\mathcal{A}}\big(r_a + \gamma\mathcal{T}_a V\big)$$

where r_a is a vector of rewards at all states for action a and $\mathcal{T}_a = P(s^{'} = i|s = j, a)$ is the transition matrix for action a. Since

$$V^*(s) = r(s,a) + \gamma E[V^*(s^{'})]$$

so $V^* = \mathcal{B}V^*$ and the optimum V^* is a fixed point of $\mathcal{B}$. Since $\mathcal{B}$ is a contraction, one can prove that the value iteration reaches V^*.

17.1.2.2 Value Iteration

In value iteration, also called backward induction, the policy π is not directly used. Instead, the value of $\pi(s)$ is calculated within $V_\pi(s)$ whenever it is needed. As a simple example of Value Iteration algorithm, we consider the Bellman optimality equation (BOE) Bellman optimality equation (BOE). In the case of the value function with BOE given in Equation (15.3.15), following Algorithm (34), the policy evaluation is given by

$$V_\pi(s) \leftarrow r(s,\pi(s)) + \gamma E_{s^{'}\sim p(s^{'}|s,\pi(s))}[V_\pi(s^{'})]$$

and the update is obtained by maximising the policy. Thus, we start with some initial function V_0^*, and compute at each time step $t = (s,a,r,s^{'})$ the estimate

$$V_{t+1}^*(s) = \max_{a\in\mathcal{A}}\big[R(s,a) + \gamma\sum_{s^{'}\in\mathcal{S}} P(s^{'}|s,a)V_t^*(s^{'})\big]$$

which converges asymptotically to V^* as $t \to \infty$.
Similarly for the Q-value function with BOE in Equation (15.3.24), starting with some initial function Q_0^*, the iterative update is given by

$$\begin{aligned} Q_{t+1}^*(s,a) &= E[r + \gamma\max_{a^{'}\in\mathcal{A}} Q_t^*(s^{'},a^{'})|s,a] \\ &= R(s,a) + \gamma\sum_{s^{'}\in\mathcal{S}} P(s^{'}|s,a)\max_{a^{'}\in\mathcal{A}} Q_{t+1}^*(s^{'},a^{'}) \end{aligned}$$

and Q_t^* will converge to Q^* as $t \to \infty$ (see Sutton et al. [2017]).
However, this algorithm is not scalable as one must compute $Q^*(s,a)$ for every state-action pair, which is computationally infeasible to compute for the entire state space. Further, we are required to know both the transition matrix of the environment and the expected next step reward. Some solutions are discussed below.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.1.2.3 Monte Carlo method

One solution to the above problem is to consider the Monte Carlo method, which does not need to know explicitly the transition probabilities. It is based on independent simulations of processes (S, A). Each process generated is called an episode. It is an offline method where the Q-value function is updated as soon as the episode is finished. We can also build an online method by letting the Q-value function be updated as soon as a new state occur.
Choose a time horizon $T \in \mathbb{N}$ such that $\sum_{t=T+1}^{\infty} \gamma^t r(s_t, a_t)$ is small enough. For each pair (s, a), generate independently M episodes $(S_s^{(j)}, A_a^{(j)})$ under the policy π. For each pair (s, a), set

$$Q_M = \frac{1}{M} \sum_{j=1}^{M} \big(r(s,a) + \gamma r(s_1^{(j)}, a_1^{(j)}) + \cdots + \gamma^T r(s_T^{(j)}, a_T^{(j)})\big)$$

According to the law of large numbers, Q_M converges to Q_π if M and T are large enough.

17.1.2.4 TD learning

Another solution to the above problem is to consider on-line learning rules called Temporal Difference (TD) learning methods (see Section (16.1)). It is a class of incremental learning procedures specialising in prediction problems which is driven by the error (or difference) between temporally successive predictions. It is a model-free approach where learning does not try to explicitly learn the environment statistics. Several TD algorithms have been proposed (see Samuel [1959], Hampson [1983], Barto et al. [1983], Holland [1986]). Sutton [1988] proved their convergence and optimality for special cases and related them to supervised learning methods.
It is an on-policy learning algorithm where the agent interacts with the environment and updates the policy based on actions taken. Each process (S, A) is called an episode. The episodes must be easily generated. Choose a time horizon T and generate M episodes (S, A). The sate-action value function is updated based on the information received. At each time step t, we assume no information is given about the environment and the sequence $s_t, a_t \to r_{t+1}, s_{t+1}$ is all the information known. That is, it does not make any assumption on the true data that generated the observation $\big(s_t, a_t, r_t, s_{t+1}\big)$.

17.1.2.4.1 Description Given the value function in Equation (15.3.14), for one episode, a naive estimate would be to replace the full value iteration algorithm

$$V_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(s,a)\big[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_t(s')\big]$$

with the approximation

$$V_{t+1}(s_t) = r_{t+1} + \gamma V_t(s_{t+1})$$

Given $V_\pi(s)$ in Equation (15.3.13), as $t \to \infty$, one can show that the estimate converges. However, even though this algorithm will improve the estimate of V on average, we can not control its variance. One way around is to introduce a learning rate parameter $\alpha_t > 0$ to produce small steps.
First, we let the TD error be

$$\sigma_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

which corresponds to the Advantage Function. This error corresponds to the error in the $TD(0)$ procedure discussed in Section (16.1) with the cost replaced by the reward. Taking the expectation of the error signal, we get

$$E[\sigma_t] = V_\pi(s) - V_t(s_t)$$

which is an instantaneous estimate of the deviation of the current estimate of V from its true value.
Then, the update equation of the estimate is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$V_{t+1}(s_t) = V_t(s_t) + \alpha_t \sigma_t$$

which we can rewrite as

$$V_{t+1}(s_t) = (1 - \alpha_t)V_t(s_t) + \alpha_t\big(r_{t+1} + \gamma V_t(s_{t+1})\big)$$

Hence, it corresponds to the stochastic approximation (see Appendix 15.3.4.3). The learning rate α determines the extent to which newly acquired information overrides old information. Setting $\alpha = 1$ would make the agent only consider the most recent information. The implementation is given in Algorithm (36).

Algorithm 36 TD-Learning Algorithm

Require: step size $\alpha \in (0, 1]$, the policy π to be evaluated
Require: Initialise step of the episode $k = 0$
Require: Initialise $V(s)$, for all $s \in \mathcal{S}$, arbitrarily except that $V(N) = 0$

```
for episode = 1 to M do
    repeat
        a ← action given by π for s
        Perform action a; (s', r) ← new state and reinforcement
        σ = r + γV(s') − V(s)
        V(s) ← V(s) + ασ
        s ← s'
        k = k + 1
    until k ⩾ N
end for
```

17.1.2.4.2 Model assumptions As discussed in Section (16.1.3), the Adaptive Heuristic Critic model of Sutton [1984] (AHC-model) is best understood as using the linear TD method for predicting discounted cumulative outcomes. In that setting, the prediction function P_t is the linear form $w \cdot x_t$, where w is the weight vector and the vector x_t is assumed to be real-valued measurements or features. The learning rule for the $TD(\lambda)$ (see Equation (16.1.17)) is given by

$$\Delta w_t = \alpha \sigma_t \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$$

where $\nabla_w P_k = x_k$. The heuristic reinforcement signal σ_t is given in Equation (16.1.6), which is the TD error σ_t above. However, in the TD learning, the latter is multiplied by the learning rate α and then used to update the prediction $V(s)$. The simplest update rule is

$$V_t \leftarrow V_t + \alpha_t \sigma_t$$

which is computed once we have transitioned to s_{t+1} and received r_{t+1}. Note, the eligibility (or trace) $\sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$ has disappeared. It has been called $TD(0)$ or one-step TD, but even in that case there is no gradient. It is argued that when the prediction function is a neural network, we update V_t by finding the output gradient with respect to the weights. This step is not captured in the above equation.

17.1.2.4.3 The SARSA model Alternatively, we can update the Q-value function. In that case, we get the SARSA (state-action-reward-state-action) (see Rummery et al. [1994]). A naive update would be

$$Q_{t+1}(s_t, a_t) = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1})$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

but it will generate a lot of variance. Hence, we consider

$$\sigma_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$$

and the update rule is

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \sigma_t$$

which we can rewrite as

$$Q_{t+1}(s_t, a_t) = (1 - \alpha_t) Q_t(s_t, a_t) + \alpha_t \big(r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1})\big)$$

Note, a_{t+1} is the action taken according to policy π, hence it is an on-policy. This algorithm converges almost surely to the optimal policy if we use the policy iteration in Algorithm (34), and if all the states and actions are sufficiently explored (infinitely explored if $M \to \infty$).

17.1.2.5 Fitted value iteration

17.1.2.5.1 The value function As discussed above, we have assumed that the transition matrix of the environment and the expected next step reward were known. If we do not know the transition dynamics, we can fit $V_\pi(s)$ to a model by using samples. To do so, we consider a function approximator to estimate the value function $V(s)$, such as neural networks (see Remark (10.1.1)). The method uses a function approximator to estimate the value function

$$V(s; \phi) \approx V^*(s)$$

where ϕ is the function parameters (or weights). Here, V^* is given either by the BOE in Equation (15.3.15), or, if we do not know the transition dynamics, from the Q-value function, by Equation (15.3.23). That is, we consider a neural network representation of value functions and back propagate the error. The loss function is given by

$$\mathcal{L}(\phi) = \frac{1}{2} \| V_\phi(s) - \max_a Q_\pi(s, a) \|^2$$

We consider dataset $\{(s_i, a_i, s_i^{'}, r_i); i = 1, ..., N\}$ generated by using some policy. The fitted value iteration algorithm is as follows:

Algorithm 37 Fitted Value Iteration Algorithm

1: Set $y_i \leftarrow \max_{a_i} \big(r(s_i, a_i) + \gamma E[V_\phi(s_i^{'})]\big)$
2: Set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \| V_\phi(s_i) - y_i \|^2$
3: Repeat step [1].

We forget the policy and directly compute the value.

17.1.2.5.2 The Q-value function In the case of the Q-value function, from Equation (15.3.20), we get the Q-Iteration algorithm:

Algorithm 38 Fitted Q-Value Iteration Algorithm

1: Set $y_i \leftarrow r(s_i, a_i) + \gamma E[V_\phi(s_i^{'})]$.
2: Set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \| Q_\phi(s_i, a_i) - y_i \|^2$.
3: Repeat step [1].

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Again, if we do not know the transition dynamics, we can try the max trick where we use the approximation $E[V_\phi(s_i^{'})] \approx \max_{a'} Q_\phi(s_i^{'}, a^{'})$. However, convergence is not guaranteed for non-linear function approximation.
The full fitted Q-Iteration algorithm is

1. Collect dataset $\{(s_i, a_i, s_i^{'}, r_i); i = 1, ..., N\}$ using some policy.
2. Repeat K times
   (a) Set $y_i \leftarrow r(s_i, a_i) + \gamma \max_{a_i^{'}} Q_\phi(s_i^{'}, a_i^{'})$.
   (b) Set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$.
3. Repeat step [1].

This algorithm is off-policy: given the state s and action a, the transition is independent of π. It approximates the value of $\pi^{'}$ at $s_i^{'}$. The error is computed as

$$\epsilon = \frac{1}{2} E_{(s,a)\sim\beta}[Q_\phi(s, a) - \big(r(s, a) + \gamma \max_{a^{'}} Q_\phi(s^{'}, a^{'})\big)]$$

If $\epsilon = 0$, we get $Q_\phi(s, a) = r(s, a) + \gamma \max_{a'} Q_\phi(s^{'}, a^{'})$, which is an optimal Q-function corresponding to the optimal policy $\pi^{'}$.
An alternative solution is the online Q-Iteration algorithm (see Ernst et al. [2005]):

Algorithm 39 Fitted Q-Value Iteration Algorithm 2

1: Take some action a_i and observe $(s_i, a_i, s_i^{'}, r_i)$.
2: Set $y_i \leftarrow r(s_i, a_i) + \gamma E[V_\phi(s_i^{'})]$.
3: Set $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)\big(Q_\phi(s_i, a_i) - y_i\big)$ where $\widehat{y}_i = Q_\phi(s_i, a_i)$.
4: Repeat step [1].

This is the Q-learning and its variants, which we will detail in Section (17.1.3).
Note:

- The observations in step [1] can be correlated. The sequential states are strongly correlated.
- The update in step [3] is not equivalent to the gradient descent since there is no gradient through the target value y_i.

***Remark* 17.1.1** *Q-learning converges for tabular value function, but not for non-tabular value function.*

17.1.3 The Q-learning algorithms

17.1.3.1 Q-learning

As discussed in Section (16.1.4), the Q-learning algorithm is a model-free off-line policy algorithm based on learning the optimal policy (see Watkins [1989] and Watkins et al. [1992]). It evaluates and improves a policy which is different from the one that generated the data. It converges directly to the optimal Q-function Q^*. One has some estimate of the Q-function $\{Q_t(s, a)\}$ which changes as new information is revealed. It solves iteratively the Bellman equation (see Equation (15.3.24)), while avoiding the transition probabilities. Similarly to the TD learning algorithm, it is based on estimating the mean with the stochastic approximation method (see Appendix (15.3.4.3)).
Choose a time horizon T and generate M episodes (S, A). Given the data point $\big(s_t, a_t, r_t, s_{t+1}\big)$, the error signal is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\sigma_t = r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q_t(s_{t+1}, a') - Q_t(s_t, a_t) \tag{17.1.2}$$

Given the optimal value $Q^*(s,a)$ in Equation (15.3.24), and taking the expectation of the error signal, we get

$$E[\sigma_t] = Q^*(s,a) - Q_t(s_t, a_t)$$

so that if the estimate is optimal, the expected value of the error signal is null.
The iterative update is given by

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \sigma_t \tag{17.1.3}$$

where $\alpha_t \in [0,1]$ is the learning factor. We can rewrite it as

$$Q_{t+1}(s_t, a_t) = (1 - \alpha_t) Q_t(s_t, a_t) + \alpha_t \Big(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q_t(s_{t+1}, a') \Big)$$

which is the one-step Q-learning in Equation (16.1.21). Note, it corresponds to the stochastic approximation method presented in Section (15.3.4.3).
As discussed above, if the estimate is the optimal Q, then Q_t should be equal to the average of the future reward:

$$Q_t(s_t, a_t) = r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q_t(s_{t+1}, a')$$

so that the update becomes

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t)$$

as expected. That is, Q^* is a fixed point (see Appendix (23.2.1.8)).
Note, a' in Equation (17.1.2) represents all actions probed in state s_{t+1}, hence it is an off-policy. It will find the optimal policy independently of the policy used during exploration, however this is true only when you visit the different states enough times. In a simulator, where every action is selected in each state with the same probability, Watkins [1989] proved that as t tends to infinity, the algorithm converges to the unique solution of the BOE, solving the problem (see Section (16.1.4)). We get the following convergence (see Gosavi [2014]):

Theorem 17.1.1 *For a small enough α, or for $\alpha \to 0$ as $t \to \infty$:*

$$\lim_{t \to \infty} Q_t(s_t, a_t) = Q^*(s,a)$$

This convergence is independent of the policy, provided that we have an exhaustive sampling of state-action pairs (s,a). According to Mohri et al. [2012], if all the states and actions are sufficiently explored (infinitely explored if $M \to \infty$) and, if some usual stochastic conditions hold, then Q converges almost surely to Q^*. Once we know the optimal Q^* we also know the optimal value function V^* and the optimal policy π^*. This is an implicit way of learning the optimal policy.

17.1.3.2 A generalisation

When learning the Q-function in Equation (17.1.3), we need to store the current predictions at each state for each action, and update them when new information comes in. This is impractical for large state-spaces and infeasible for continuous state-spaces. We therefore rely on function approximation to generalise prediction between states. One solution, among others, is to use neural networks (see Remark (10.1.1)).
In order to obtain a generalisation of RL for large state-spaces, Lin [1993] considered four techniques to speed up RL:

- generalisation by neural networks

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- experience replay
- using action models
- teaching

These techniques were combined with AHC-learning (see Sutton [1984]) and Q-learning (see Watkins [1989]), resulting in eight RL frameworks named:

- AHCON: connectionist AHC-learning
- AHCON-R: AHCON plus experience replay
- AHCON-M: AHCON plus using action models
- AHCON-T: AHCON plus experience replay plus teaching
- QCON: connectionist Q-learning
- QCON-R: QCON plus experience replay
- QCON-M: QCON plus using action models
- QCON-T: QCON plus experience replay plus teaching

Here, the Q-function is represented by neural networks and learned by temporal difference and backpropagation. That is,

1. the temporal difference compute the error between temporally successive predictions
2. backpropagation minimises the error by modifying the weights of the networks

The author used back-propagation neural networks to store information learnt by the Q-learning algorithm and proposed backward replay. The algorithm stores the state-action pairs visited by the system until the end of a trial when the rewards are known, and then replay them in reverse order to train the network. We can either use a single network with $|\mathcal{A}|$ outputs, or $|\mathcal{A}|$ separate networks, each with a single output. Lin chose the latter.

17.1.3.2.1 The AHCON It is a connectionist implementation of AHC-learning consisting in three components:

1. an evaluation network called V-net
2. a policy network called policy-net
3. a stochastic action selector

The framework decomposes RL into two (concurrent) learning processes (subtasks):

1. Learning a V-function: Using TD methods, construct a V-net that models the V-function.
2. Learning a policy: Adjust the policy-net so that it will assign higher merits to actions that result in higher utilities as measured by the V-net.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The policy is to choose the action having the highest merit. Both processes interact closely: a change in the policy redefine the target V-function, and an update to the V-function causes the policy to change. Thus, this interaction can either prevent convergence or stabilise to a suboptimal policy.
In [1], the author considered the $TD(0)$ method (see Section (16.1)) and defined the optimal V-function as

$$V(s) = r + \gamma V(s^{'})$$

where s is the current state and $s^{'}$ is the next state. The implementation is given in Algorithm (40).

Algorithm 40 AHCON-Algorithm

Require: Initialise $k = 0$

1: **repeat**
2: $x \leftarrow$ current state; $e \leftarrow V(s)$
3: $a \leftarrow select(policy(s), T)$;
4: Perform action a; $(s^{'}, r) \leftarrow$ new state and reinforcement
5: $e^{'} = r + \gamma V(s^{'})$
6: Adjust the V-net by backpropagating $TD(0)$ error $\sigma = (e^{'} - e)$ through it with input s.
7: Adjust the policy-net by backpropagating error Δ through it with input s, where

$$\Delta_i = \begin{cases} e^{'} - e \text{ if } i = a\,, i \in \mathcal{A} \\ 0 \text{ otherwise} \end{cases}$$

8: $k = k + 1$
9: **until** $k \geqslant N$

The difference between the two sides of the V-net equation is the TD error σ. It is reduced by adjusting the weights of the V-net by using the backpropagation algorithm (Step 6). The policy-net is also adjusted (Step 7) according to the same TD error. That is,

- if $e^{'} > e$, action a is better than previously expected, so that the policy is modify to increase the merit of the action.
- if $e^{'} < e$, the inverse is true, and the policy is modified to decrease the merit.

17.1.3.2.2 The QCON It is a connectionist implementation of Q-learning where the Q-function is estimated by a network called Q-net. The policy is to choose the action a such that $Q(s, a)$ is maximal over all actions. Hence, the Q-net is both an evaluation function and a policy. The implementation is given in Algorithm (41).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 41 QCON-Algorithm

Require: Initialise $k = 0$

1: **repeat**

2: $x \leftarrow$ current state; for each action $i \in \mathcal{A}$, $U_i \leftarrow Q(s, i)$;

3: $a \leftarrow select(U, T)$;

4: Perform action a; $(s^{'}, r) \leftarrow$ new state and reinforcement

5: $u^{'} = r + \gamma \max_{k \in \mathcal{A}} Q(s^{'}, k)$

6: Adjust the Q-net by backpropagating error ΔU through it with input s, where

$$\Delta U_i = \begin{cases} u^{'} - U_i \text{ if } i = a \text{ , } i \in \mathcal{A} \\ 0 \text{ otherwise} \end{cases}$$

7: $k = k + 1$

8: **until** $k \geqslant N$

In this algorithm, the TD error σ is given by Equation (17.1.2) as

$$\sigma = r + \gamma \max_{k \in \mathcal{A}} Q_t(s^{'}, k) - Q(s, a)$$

and the neural network weights w_t are updated as follows:

$$\Delta w_t = \eta \sigma \nabla_w Q_t$$

where η is the learning rate, and $\nabla_w Q_t$ is a vector of the output gradients $\frac{\partial Q_t}{\partial w_t}$ calculated by backpropagation in Step 6. Multiple networks are use to represent the Q-function; one network for each action and each with a single output. This is the OAON (one action one network).

17.1.3.2.3 Experience replay Both the AHC and Q-learning above are inefficient because the experiences obtained by trial and error are only used once to adjust the networks and then discarded. However, some experiences may be rare and others costly to obtain. The author proposed to reuse the experiences in an effective way called experience replay. Here, the learning agent remembers its past experiences and repeatedly presents them to its learning algorithm as if the agent was re-experiencing them again. As a result:

- the process of credit/blame propagation is sped up, and
- the agent gets a chance of refreshing its past experiences.

Simply described in a scenario with two parts A and B. While the agent is exploring A, he might have forgotten part B. Experience replay allows the agent to re-experience part B. Note, experience replay can be more effective in propagating credit/blame if a sequence of experiences is replayed in temporally backward order. This was noticed by Sutton [1988]. See discussion in Section (16.1.3.3) in the case of a random walk. It can be even more effective if $TD(\lambda)$ methods are used with $\lambda > 0$. In that case, the TD error for adjusting network weights is determined by the discrepancy between multiple consecutive predictions.
Presenting an experience to the algorithm means to bind the variables $(s, a, s^{'}, r)$ used in the algorithm with the experience. In AHCON-R, the author repeat presentation of past experiences to the algorithm for ACHON. However, the experiences involving non-policy actions (according to the current policy) are not presented. Similarly, QCON-R repeats presentation of past experiences to the algorithm for QCON, also only replaying the actions following the current policy. That is, both AHCON-R and QCON-R only replay policy actions. Further, to gain efficiency in computational time and memory space, they only replay recent experiences since experiences in far past may involve bad action choices that we do not want to refresh.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The learning assume that the next states are visited in proportion to their probabilities, which can be invalidated by improper experience replay, causing an over-replay problem. One solution is to replay each experience roughly the same number of times except that we exclude replaying bad actions.

17.1.3.2.4 Using $TD(\lambda)$ Watkins [1989] proposed to combine Q-learning with TD-learning (see Section (16.1)) so that the current update error is used to adjust both the current estimate of Q_t and that of previous states, by keeping a weighted sum of earlier error gradients. It is given by

$$\Delta w_t = \eta \sigma_t \sum_{k=0}^{t} (\lambda\gamma)^{t-k} \nabla_w Q_k$$

where σ_t is the TD error in Equation (17.1.2), λ weights the relevance of the current error on earlier Q-value predictions. Hence, setting $\lambda = 0$ we recover the one-step Q-learning equation. Note, this is not a true TD-learning algorithm unless the greedy policy is followed. That is, the action $\max_{a'} Q(s_{t+1}, a')$ must be performed at every time step.
Lin [1993] proceeded differently. After performing simulation experiments, he found that the best model was QCON-T, which combines Q-learning, neural networks, experience replay, and teaching. He then extended the experience replay technique to use $TD(\lambda)$ methods for general λ. The results indicated a speed advantage in using nonzero λ value.
We consider the sequence of state transitions:

$$(s_0, a_0, s_1, r_0) \cdots (s_n, a_n, s_{n+1}, r_n)$$

Let Δ_t^0 be the $TD(0)$ error, and Δ_t^λ be the $TD(\lambda)$ error. Let U_t be the predicted utility of state s_t. Then, Δ_t^0 and Δ_t^λ are given by

$$\begin{aligned} \Delta_t^0 &= r_t + \gamma U_{t+1} - U_t \\ \Delta_t^\lambda &= \sum_{k=0}^{n-t-1} (\gamma\lambda)^k \Delta_{t+k}^0 \end{aligned}$$

We can rewrite Δ_t^λ as follows:

$$\begin{aligned} \Delta_t^\lambda &= \Delta_t^0 + \sum_{k=1}^{n-t-1} (\gamma\lambda)^k \Delta_{t+k}^0 \\ &= \left[r_t + \gamma U_{t+1} + \sum_{k=1}^{n-t-1} (\gamma\lambda)^k \Delta_{t+k}^0\right] - U_t \end{aligned}$$

Then, the $TD(\lambda)$ return, which is the estimated utility from state s_t, is

$$R_t^\lambda = r_t + \gamma U_{t+1} + \sum_{k=1}^{n-t-1} (\gamma\lambda)^k \Delta_{t+k}^0$$

The $TD(\lambda)$ error is given by

$$\Delta_t^\lambda = R_t^\lambda - U_t$$

We can rewrite the above equation for R_t^λ as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} R_t^\lambda &= r_t + \gamma U_{t+1} + \gamma\lambda\Delta_{t+1}^\lambda \\ &= r_t + \gamma U_{t+1} + \gamma\lambda\big(R_{t+1}^\lambda - U_{t+1}\big) \\ &= r_t + \gamma\big[(1-\lambda)U_{t+1} + \lambda R_{t+1}^\gamma\big] \end{aligned}$$

which is the basis of the experience replay algorithm. When $0 < \lambda < 1$, the TD return is some aggregate of the predicted return ($TD(0)$ return) and the actual return ($TD(1)$ return). Since TD methods train the function using TD returns as target outputs, $TD(1)$ is a supervised learning method.
To compute TD returns in the AHC-learning, we set

$$U_{t+1} = V(s_{t+1})$$

In the case of Q-learning, we set

$$U_{t+1} = \max_{a^{'}\in\mathcal{A}} Q(s_{t+1}, a^{'})$$

When the V-function and Q-function are implemented with neural networks, we use the backpropagation algorithm to reduce the $TD(\lambda)$ error.

Algorithm 42 QCON-Replay-Algorithm

Require: $\{(s_0, a_0, s_1, r_0)\cdots(s_n, a_n, s_{n+1}, r_n)\}$
Require: Initialise $k = 0$
1: $t \leftarrow n$
2: $e_t \leftarrow Q(s_t, a_t)$;
3: $u_{t+1} \leftarrow \max_{k\in\mathcal{A}} Q(s_{t+1}, k)$
4: $e_t^{'} \leftarrow r_t + \gamma\big[(1-\lambda)u_{t+1} + \lambda e_{t+1}^{'}\big]$
5: Adjust the Q-net for action a_t by backpropagating error $(e_t^{'} - e_t)$ through it with input s_t.
6: If $t = 0$ exit; else $t \leftarrow t-1$ go to 2.

The author implemented the backpropagating equation (see Equation (16.1.15)) as follows:

$$\Delta w_t = \eta\big(Q_t^{'} - Q_t\big)\nabla_w Q_t$$

where

$$Q_t^{'} = r_{t+1} + \gamma\big[(1-\lambda)\max_{a^{'}\in\mathcal{A}} Q_{t+1} + \lambda Q_{t+1}^{'}\big]$$

Here, each $Q_t^{'}$ depends recursively on future $Q_t^{'}$ values, so that updating can only occur at the end of each trial. Hence, until then, all state-action pairs must be stored and then presented in a temporally backward order to propagate the errors. This is the backward replay.
Recall, Δ_t^λ is some weighted sum of $\Delta_t^0, \Delta_{t+1}^0, ..., \Delta_{t+k}^0, ...$ That is, the contribution of Δ_t^0 decays exponentially by the temporal distance between state s_t and the state to which the learning change is applied. Let D be the temporal distance for which this contribution is decayed to $\frac{1}{e} \approx 0.37$ of the original amount, that is, $\frac{1}{e} = \lambda^D$. Then,

$$D = \frac{-1}{\log_e(\lambda)}$$

The larger the λ value, the farther credits can be propagated from goal states to preceding states at a time. However, large λ may not be desirable. One strategy is to start the learning with $\lambda = 1$ and gradually reduce it to 0. Sutton [1988] found that $\lambda = 1$ gave the worst results and that $\lambda = 0.3$ was a good choice.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.1.3.2.5 The modified QCON Rummery et al. [1994] considered on-line algorithms and showed convergence to optimal policies over a large range of parameters. This way, we do not have to store the state-action pairs. They introduced the Modified Connectionist Q-learning and extended the $Q(\lambda)$ by using continuous function approximators.

17.1.3.3 Neural fitted Q-value

We have introduced in Section (17.1.2.5) the neural fitted value function and Q-value function. Following this approach, the above Q-learning rule in Equation (17.1.2) can be implemented in a neural network. In that case, an error function measuring the difference between the current Q-value and the new value is assigned. One can then apply a gradient descent technique such as backpropagation to adjust the weights on the network to minimise the error.
In an on-line update rule, the update is applied after each new sample. This requires generating a very large number of episodes to get a near optimal policy. The reason being that adjusting weights for a particular state-action pair leads to unpredictable changes occurring elsewhere in the state-action space. Riedmiller [2005] proposed the neural fitted Q-iteration (NFQ), where the update is performed off-line by considering an entire set of transition experiences. They are collected in triples of the form $(s, a, s^{'})$ in a sample set $\mathcal{D}$. This way, advanced supervised learning methods can be used, that converge faster and are more reliable than on-line gradient descent.
The NFQ consists of two steps: (1) The generation of the training set P, and (2) the training of these patterns within a multi-layer perceptron. For iteration l, the input is the pair (s_l, a_l). The target value is made of the cost $c(s_l, a_l, s_{l+1})$ and the expected minimal path cost for the next state $s_l^{'}$. See Algorithm (43).

Algorithm 43 NFQ-Algorithm

Require: Input: a set of transition samples D
Require: Output: Q-value function Q_N
Require: Initialise $k = 0$, $initMLP() \rightarrow Q_0$
1: **repeat**
2: generatePatternSet $P = \{\big((s_l, a_l), \mu_l\big), l = 1, ..., \#D\}$ where

$$\mu_l = c(s_l, a_l, s_l^{'}) + \gamma \min_b Q_k(s_l^{'}, b)$$

3: $PropTraining(P) \rightarrow Q_{k+1}$
4: $k = k + 1$
5: **until** $k \geqslant N$

Hafner et al. [2011] extended the approach by proposing the neural fitted Q-iteration with continuous actions.

17.1.3.4 Deep Q-learning

As discussed in the above section, when the state-action space is very large, we can not store each Q-factor separately (tabular function), and must rely on using a function approximator to estimate the Q-value function $Q(s, a)$, such as neural networks (see Remark (10.1.1)). In that case, the Q-learning uses a function approximator (Q-network) to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

where θ is the function parameters (or weights). Ernst et al. [2005] and Riedmiller [2005] showed that one solution was to consider a neural network representation of value functions and back propagate the error. When the approximator is a deep NN, it is called deep Q-learning (see Mnih et al. [2015]). In that setting, the end-to-end reinforcement learning agent, called a deep Q-network (DQN), is able to continuously adapt its behaviour without any human intervention.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We want to find a Q-value function that satisfies the BOE in Equation (15.3.24). To do so, we generate different episodes of the process (S, A) and compute, at each iteration $t \in [1, T]$, the forward and backward pass:

1. The forward pass: Define the loss function

$$L_t(\theta_t) = E_{s,a\sim\rho(\cdot)}[\big(y_t - Q(s,a;\theta_t)\big)^2]$$

where ρ is the MDP equilibrium distribution over state s and action a, and the target value is

$$\begin{aligned} y_t &= E_{s'\sim p(s'|s,\pi(s))}[r + \gamma Q^*(s^{'}, \pi(s^{'}); \theta_{t-1})|\, s, a] \\ &= E_{s'\sim p(s'|s,\pi(s))}[r + \gamma \max_{a'\in\mathcal{A}} Q^*(s^{'}, a^{'}; \theta_{t-1})|\, s, a] \text{ (see Proposition (8))} \end{aligned}$$

which depends itself on the Q-network. Thus, the target depends on the network weights. The parameters from the previous iteration θ_{t-1} are held fixed.
We iteratively try to make the Q-value close to the target value y_t it should have if the Q-function corresponds to the optimal Q^* and optimal policy π^*. This is a calibration (tuning) process.

2. The backward pass: The gradient update (with respect to θ)

$$\nabla_{\theta_t} L_t(\theta_t) = E_{s,a\sim\rho(\cdot),s'\sim p(s'|s,\pi(s))}[\Big(r + \gamma \max_{a'\in\mathcal{A}} Q(s^{'}, a^{'}; \theta_{t-1}) - Q(s,a;\theta_t)\Big)\nabla_{\theta_t} Q(s,a;\theta_t)|\, s, a]$$

***Remark* 17.1.2** *The network $Q^*(s, \pi(s); \theta)$ being updated is also used in computing the target value, which is prone to divergence.*

For instance, in the case of Q-learning, we can choose a deep neural network (DNN) with inputs being vector representation of the state s and outputs are multiple nodes representing $Q(s, a|\theta)$. After presentation of a state s, applying a greedy policy on the output Q, we get the reward and the new state and compute the error signal (compare with Equation (17.1.2))

$$\sigma_t = r_{t+1} + \gamma \max_{a'\in\mathcal{A}} Q_t(s_{t+1}, a^{'}|\,\theta) - Q_t(s_t, a_t|\,\theta)$$

In general, we do not compute the full expectation in the above gradient, but consider stochastic gradient descent (see Section (5.1.4)). The weights are updated after every time step, and the expectation is replaced by single samples from ρ and $p(\cdot|s, \pi(s))$. That is, if we let σ^2 be the objective function, taking the expectation, we recover the above loss function. We can then get the gradient with respect to θ as follows:

$$\nabla_\theta \sigma^2 \sim -\sigma_t \nabla_\theta Q_t(s_t, a_t|\,\theta)$$

and the update of the parameter is given by

$$\theta^{'} = \theta - \beta_t \sigma_t \nabla_\theta Q_t(s_t, a_t|\,\theta)$$

where β_t is a learning rate. The deep Q-learning algorithm is

Algorithm 44 Deep Q-Learning Algorithm

1: Take some action a_t and observe (s_t, a_t, r_t, s_{t+1}).
2: Set $\theta \leftarrow \theta - \beta_t \sigma_t \nabla_\theta Q_t(s_t, a_t|\,\theta)$.
3: Repeat step [1].

Recall from Section (16.1.3), the asymptotic proof of convergence of the TD-procedures was performed on linear prediction functions. Introducing non-linear function approximators means that convergence is no-longer guaranteed.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.1.3.5 Experience replay

When training the above Q-network, it is problematic to learn from batches of consecutive samples because:

- Samples are correlated, leading to inefficient learning. However, we assumed that the data was independently and identically distributed.
- The current Q-network parameters determine the next training samples, which might lead to poor feedback loops.

One solution is to use an experience replay mechanism (see Lin [1993]). It is described as follows:

- Continuously update a replay memory table of transitions (agent's experiences) $e_t = (s_t, a_t, r_t, s_{t+1})$ as episodes are played. Need to periodically feed the replay buffer.
- Each transition can contribute to multiple weight updates to get greater data efficiency.
- Train the Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples.
- Use target networks $Q_{\theta'}$ when computing the target values.

Thus, one can learn value functions with function approximators in a stable and robust way by training the network

1. off-policy with samples from replay buffer to minimise correlations between samples.
2. with a target Q-network to give consistent targets during temporal difference backups.

The classic deep Q-learning algorithm (DQN) with replay buffer and target networks $Q_{\theta'}$ with parameters $\theta^{'}$ (different from θ) is as follows:

Algorithm 45 Deep Q-Learning with Replay Algorithm

1: Take some action a_t and observe $e_t = (s_t, a_t, r_t, s_t^{'})$, add it to the buffer $\mathcal{B}$.
2: Sample mini-batch $\{s_j, a_j, r_j, s_j^{'}\}$ from $\mathcal{B}$ uniformly.
3: Compute $y_j = r_j + \gamma \max_{a_j^{'}} Q_{\theta'}(s_j^{'}, a_j^{'})$ using target network $Q_{\theta'}$.
4: Update $\theta \leftarrow \theta - \beta \sum_j \nabla Q_\theta(s_j, a_j)\big(Q_\theta(s_j, a_j) - y_j\big)$
5: Update $\theta^{'}$, copy θ every N steps.
6: Repeat step [1].

The target networks $Q_{\theta'}$ are time-delayed copies of their original networks that slowly track the learned networks. It greatly improves stability in learning. This way, the networks are less interdependent.
Alternative target networks exist. For instance the step [5] can be defined as

$$\theta^{'} \leftarrow \tau\theta^{'} + (1-\tau)\theta \tag{17.1.4}$$

where τ is chosen close to 1. It is called the Polyak averaging.

We present another implementation of the deep Q-learning (see Mnih et al. [2015]) with experience replay in Algorithm (46). It let agents remember and reuse experiences from the past. The authors considered an ϵ-greedy policy that chooses either the best action with respect to the Q-network with probability $1-\epsilon$ or another random action with probability ϵ. When ϵ is close to 1, the algorithm explores as many pair (s, a) as possible. On the other hand, if ϵ is close to zero, the policy is almost a greedy policy that exploits the data to improve as much as possible the Q-value

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

function. The trade-off exploration-exploitation is managed by choosing ϵ close to 1 at the start of the training to generate diversified data to make the learning easier. Then, at the end of the training, ϵ is set close to 0 to find the optimal Q-value function faster. As an example, ϵ can decrease exponentially with respect to the training step, yielding a sequence $(\epsilon_n)_{n\in\mathbb{N}}$ defined by the relation

$$\epsilon_n = \big(\epsilon_{max} - \epsilon_{min}\big)e^{-n\Delta\epsilon} + \epsilon_{min}$$

where ϵ_{max} denotes the maximum exploration probability, ϵ_{min} denotes the minimum exploration probability, and the variation $\Delta\epsilon$ represents the exploration decay coefficient.
Rather than working with histories of arbitrary length as input to the NN, the authors considered working on fixed length representation produced by a function ϕ.

Algorithm 46 Deep Q-learning with Experience Replay Algorithm

Require: : training set S, number of training steps T
Initialise replay memory $\mathcal{D}$ to capacity N
Initialise action-value function Q with random weights
for $episode = 1$ to M **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi_1(s_1)$
  for $t = 1$ to $T - 1$ **do**
    With probability ϵ select a random action a_t, otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set

$$y_j = \begin{cases} r_j \text{ for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a^{'}; \theta) \text{ for non-terminal } \phi_{j+1} \end{cases}$$

    Perform a gradient descent step on $\big(y_j - Q(\phi_j, a_j; \theta)\big)^2$
  end for
end for

Here, the experience transition are uniformly sampled from a replay memory. Schaul et al. [2016] developed a framework for optimising experience, so as to replay important transitions more frequently and learn more efficiently. This is obtained by using a prioritised experience replay in DQN. However, this method introduces a bias.

17.1.3.6 Convergence

The Q-learning algorithm is known to overestimate the action values under certain conditions. We discuss a few solutions proposed to overcome this problem.

17.1.3.6.1 The problem To understand the argument of the double Q-learning we consider the target value

$$y_j = r_j + \gamma \max_{a^{'}} Q_{\theta'}(s_j^{'}, a_j^{'})$$

in the classic Q-learning algorithm with target networks (see Algorithm (45)), and concentrate on the term $\max_{a'} Q_{\theta'}(s^{'}, a^{'})$. Assuming we have two random variables X_1 and X_2, given the Jensen inequality in Equation (23.2.4), we have

$$E[\max(X_1, X_2)] \geqslant \max\big(E[X_1], E[X_2]\big)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Thus, $\max_{a'} Q_{\theta'}(s', a')$ overestimates the next value. In fact, we have

$$\max_{a'} Q_{\theta'}(s', a') = Q_{\theta'}\left(s', \arg\max_{a'} Q_{\theta'}(s', a')\right)$$

where the action is selected according to $Q_{\theta'}$. Thus, we need to decorrelate the noise in the latter.

17.1.3.6.2 The double Q-learning One solution is to consider the double Q-learning, introduced in a tabular setting, and generalised by van Hasselt et al [2015] to work with large-scale function approximation. The method uses two networks:

$$\begin{aligned} Q_{\theta_1}(s,a) &\leftarrow r + \gamma Q_{\theta_2}\left(s', \arg\max_{a'} Q_{\theta_1}(s', a')\right) \\ Q_{\theta_2}(s,a) &\leftarrow r + \gamma Q_{\theta_1}\left(s', \arg\max_{a'} Q_{\theta_2}(s', a')\right) \end{aligned}$$

The main idea is to hope that the two functions Q_{θ_i}, $i = 1, 2$, are noisy in different ways. We can use both the current network and the target network. The difference between the Q-learning and double Q-learning is as follows:

- In the case of the standard Q-learning we use the target network $Q_{\theta'}$. The target value is

$$y = r + \gamma Q_{\theta'}\left(s', \arg\max_{a'} Q_{\theta'}(s', a')\right)$$

- In the case of the double Q-learning we use the current network Q_θ to evaluate the action and the target network $Q_{\theta'}$ to evaluate the value. The target value is

$$y = r + \gamma Q_{\theta'}\left(s', \arg\max_{a'} Q_{\theta}(s', a')\right)$$

17.1.3.6.3 The multi-step returns Another solution is to consider off-policy, return-based reinforcement learning, and use multi-step returns (see Munos et al. [2016]). To understand the argument we consider the target value

$$y_{j,t} = r_{j,t} + \gamma \max_{a_{j,t+1}} Q_{\theta'}(s_{j,t+1}, a_{j,t+1})$$

We use the notion of actor-critic with Advantage Function given in Equation (17.1.16) and construct multi-step targets

$$y_{j,t} = \sum_{t'=t}^{t'+n-1} r_{j,t'} + \gamma^n \max_{a_{j,t+n}} Q_{\theta'}(s_{j,t+n}, a_{j,t+n})$$

where $\sum_{t'=t}^{t'+n-1} r_{j,t'}$ is the n-step return estimator. Note, we need the transitions $(s_{j,t'}, a_{j,t'}, s_{j,t'+1})$ to come from π for $t' - t < n - 1$. One can dynamically choose n to get only on-policy data, or use importance sampling.

17.1.3.7 The continuous state-action method

In the discrete state-action models considered so far, both the value function and the policy function can be stored in a tabulated form. Further, in fitted value methods, even if the DQN solves problems with high-dimensional observation spaces, it can only handle discrete and low-dimensional action spaces. However, physical control problems have continuous and high dimensional action spaces. Algorithms relying on finding the action maximising the action-value function, such as DQN, would require an iterative optimisation process at every step. In the case of continuous-state models, one has to rely on function approximations to represent compactly the value function, or the policy function, or both.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.1.3.7.1 The continuous action method In the case of continuous action, the target value in the Q-learning (see Algorithm (45)) is given by

$$y_j = r_j + \gamma \max_{a'} Q_{\theta'}(s_j^{'}, a_j^{'})$$

The problem becomes that of estimating the maximum.
Discretising the action space has many limitations. Considering the approximation

$$\max_a Q(s,a) \approx \max\{Q(s,a_1), ..., Q(s,a_n)\}$$

is easily parallelisable but not very accurate. We can also consider cross-entropy method (CEM), which is an iterative stochastic optimisation for low dimensions.
Alternatively, we can use a class of functions which is simple to optimise. For instance, the normalised advantage functions (NAF) introduced by Gu et al. [2016] is an alternative to the policy gradient (see Section (17.1.4)) and the actor-critic methods (see Section (17.1.5)). It is given by

$$Q_\theta(s,a) = -\frac{1}{2}\big(a - \mu_\theta(s)\big)^\top P_\theta(s)\big(a - \mu_\theta(s)\big) + V_\theta(s)$$

where

$$\arg\max_a Q_\theta(s,a) = \mu_\theta(s) \text{ and } \max_a Q_\theta(s,a) = V_\theta(s)$$

Another solution is to learn an approximator maximiser. Lillicrap et al. [2016] considered an off-policy actor-critic, model-free algorithm based on the deterministic policy gradient (DPG) (see Section (17.1.6.1)) that can operate over continuous action spaces. It is called the DDPG (deep deterministic policy gradient). Recall, we can express the maximum as

$$\max_a Q_\theta(s,a) = Q_\theta\big(s, \arg\max_a Q_\theta(s,a)\big)$$

The main idea is to approximate the deterministic policy $\pi(s) = \arg\max_a Q_\theta(s,a)$ with a function. In the same spirit as the double Q-learning, we can learn the deterministic policy by training another network $\mu_\phi(s)$, with parameters ϕ, such that $\mu_\phi(s) \approx \arg\max_a Q_\theta(s,a)$. The actor function μ_ϕ maps the states to a specific action. The critic $Q_\theta(s,a)$ is learned by using the Bellman equation as in Q-learning. The actor is updated by applying the chain rule to the expected return from the start distribution J with respect to the actor parameters ϕ:

$$\begin{aligned}\nabla_\phi J &\approx E_{s\sim\rho}[\nabla_\phi Q_\theta(s,a)\big|_{s=s_t, a=\mu(s_t|\phi)}] \\ &= E_{s\sim\rho}[\nabla_a Q_\theta(s,a)\big|_{s=s_t, a=\mu(s_t|\phi)} \nabla_\phi \mu(s|\phi)\big|_{s=s_t}]\end{aligned}$$

The actor update satisfies

$$\phi \leftarrow \arg\max_\phi Q_\theta(s, \mu_\phi(s)) \text{ where } \frac{dQ_\theta}{d\phi} = \frac{dQ_\theta}{da}\frac{d\mu_\phi}{d\phi}$$

Similarly to the Q-learning algorithm with target networks $Q_{\theta'}$ (see Algorithm (45)), the new target becomes

$$y_j = r_j + \gamma Q_{\theta'}(s_j^{'}, \mu_\phi(s_j^{'}))$$

where the deterministic policy is estimated with the actor function μ_ϕ.
Rather than directly copy the weights when updating the networks, the authors considered a soft target update. They created a copy of the actor and critic networks, $Q_{\theta'}(s,a)$ and $\mu_{\phi'}(s)$, respectively, that are used to compute the target values. The parameters (weights) are slowly updated with Polyak averaging (see Equation (17.1.4)). Further, to fully

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

explore the continuous action space, the authors split the problem of exploration from that of learning and considered the policy

$$\mu_{\phi'}(s_t) = \mu_\phi(s_t) + \mathcal{N}$$

where $\mathcal{N}$ is a noise process. They considered an Ornstein-Uhlenbeck process to generate temporally correlated exploration.
The algorithm to learn an approximate maximiser is:

Algorithm 47 DDPG Algorithm

Randomly initialise the critic network $Q_\theta(s,a)$ and the actor $\mu_\phi(s)$ with weights θ and ϕ.
Initialise target network $Q_{\theta'}$ and $\mu_{\phi'}$ with weights $\theta^{'} \leftarrow \theta$ and $\phi^{'} \leftarrow \phi$.
Initialise replay buffer $\mathcal{B}$.
for $episode = 1$ to M **do**
  Initialise random process $\mathcal{N}$ for action exploration. Receive initial observation state s_1.
  for $t = 1$ to T **do**
    Take some action $a_t = \mu_\phi(s_t) + \mathcal{N}_t$ and observe $(s_t, a_t, s_t^{'}, r_t)$, add it to the batch $\mathcal{B}$.
    Sample mini-batch $\{s_j, a_j, s_j^{'}, r_j\}$ from $\mathcal{B}$ uniformly.
    Compute $y_j = r_j + \gamma Q_{\theta'}(s_j^{'}, \mu_{\phi'}(s_j^{'}))$ using the target network $Q_{\theta'}$ and $\mu_{\phi'}$.
    Update the critic $\theta \leftarrow \theta - \alpha \sum_j \nabla_\theta Q_\theta(s_j, a_j)\big(Q_\theta(s_j, a_j) - y_j\big)$.
    Update the actor $\phi \leftarrow \phi + \beta \sum_j \frac{dQ_\theta}{da}(s_j, a)\frac{d\mu_\phi}{d\phi}(s_j)$.
    Update the target networks $\theta^{'}$ and $\phi^{'}$ with Polyak averaging.
  end for
end for

17.1.3.7.2 The continuous state-action method One solution to the continuous state-action is to use linear architectures expressing the function as a linear combination

$$f(x) = \sum_k \theta_k \phi_k(x)$$

for some fixed basis functions $\{\phi_k(x); k = 1, ..., K\}$. It replaces the problem of general function approximation in a non-parametric setting by a linear parametrised function approximation, making it easier to solve as we only need to fit the parameters $\{\theta_k\}$. However, it is not an easy task to find a good set of basis functions as well as the optimal number of such functions. This method has been used in finance in the pricing of American options with Monte Carlo. This is the Least Square Monte Carlo (LSMC) method (see Longstaff et al. [2001]). Using similar techniques, the Q-learning was extended to continuous state-action cases in Fitted Q-Iteration (FQI) method (see Ernst et al. [2005], Murphy [2005]). Note that the discrete-state case can be considered as a special case of the continuous-state formulation.

17.1.4 The policy gradient methods

In the previous sections, we have presented methods that estimate the Q-value functions and directly use them to select the actions without an explicit policy structure. While simple, these methods are prone to convergence failure. Another problem with Q-learning (see Section (17.1.3)) is that even though the Q-function may be very complex, the policy can be much simpler. So, we could directly learn the latter from a collection of policies. The classical form of policy iteration performs the policy evaluation in one step by solving a Bellman equation for the given policy called the Poisson equation. However, this step requires the transition probabilities, which might not be known. We can then consider a modified policy iteration where we use a Q-factor version of the Poisson equation, and apply a Q-learning like approach to solve the equation (see Section (17.1.2)).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Alternatively, we can let the policy be explicitly represented by its own function approximator, and update it according to the gradient of expected reward with respect to the policy parameters. This is called policy improvement. The main idea is to represent the policy by a parametric probability distribution $\pi_\theta(s,a) = P(a|\ s;\theta)$ that stochastically selects action a in state s according to parameter θ. One solution is to sample this policy and adjust the policy parameters in the direction of greater cumulative reward. We can directly learn the policy with a neural network, since they are function approximator (see Remark (10.1.1)), and use an independent estimate of the value of a policy for an update rule. The REINFORCE algorithm (see Williams [1992]) and actor-critic methods (see Sutton et al. [2000]) are examples of this approach.

To implement a policy gradient method, we first define a class of parametrised policies as

$$\prod = \{\pi_\theta, \theta \in \mathbb{R}^m\}$$

where θ is the policy parameters. As an example of a parametric policy, the Gibbs distribution (see Appendix (24.6.3.1)) in a linear combination of features is given by

$$\pi_\theta(s,a) = \frac{e^{\beta\theta^\top f(s,a)}}{\sum_{a'\in\mathcal{A}} e^{\beta\theta^\top} f(s,a')}\ , \forall s \in \mathcal{S}\ , a \in \mathcal{A}$$

where f is some d-dimensional feature basis for representation of the pairs (s,a). In order to learn θ we need an objective function for π. For each policy, we define the value function as

$$J(\theta) = E[\sum_{t\geqslant 0} \gamma^t r_t|\pi_\theta] \tag{17.1.5}$$

We want to find the optimal policy

$$\theta^* = \arg\min_\theta J(\theta)$$

One solution is to use the gradient ascent on policy parameters θ. It leads to the REINFORCE algorithm, which we have described in Section (15.2) in the case of stationary reward, where $< r >= R(W)$. In presence of stochastic reward, one can use some Monte Carlo (MC) (see Appendix (26.2)) policy gradient methods (see Peters et al. [2006]) where the objective function can be approximated as

$$J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t\geqslant 0}\gamma^t r_{t,i}$$

with N episodes (see Section (17.1.2.3)). They are on-policy methods directly maximising the cumulative future returns with respect to the policy. However, unless control variate, or other variance reduction methods (see Appendix (26.2.2)), is applied, MC simulations suffer from high variance. We distinguish between methods using full MC returns, such as REINFORCE (see Williams [1992]), and methods with function approximation such as actor-critic methods (see Sutton et al. [2000]) which optimise the policy against a critic. In the latter, the gradient is estimated with the help of an approximate Q-value function or Advantage Function (the critic).

***Remark* 17.1.3** *Gradient-based algorithms can not be proved to converge to a globally optimal policy. We can only hope to prove convergence to a local maximum of $J(\theta)$.*

17.1.4.1 The REINFORCE algorithm

We use the notation in Section (15.3.3), where τ is a trajectory and $p(\tau)$ is the joint mass function. Considering episodic tasks, we set $\gamma = 1$ in Equation (15.3.10), and denote

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$R(\tau) = \sum_{t \geqslant 0} r(s_t, a_t)$$

the total reward for the trajectory $\tau = (s_0, a_0, r_0), (s_1, a_1, r_1), ...$ over $\mathcal{S} \times \mathcal{A} \times \mathbb{R}$. In that setting, we can write the expected reward (objective) as

$$\begin{aligned} J(\theta) &= E_{\tau \sim p(\tau;\theta)}[R(\tau)] \\ &= \int_\tau R(\tau) p(\tau;\theta) d\tau \end{aligned}$$

where $p(\cdot;\theta)$ is the density function of that trajectory with policy parameters θ (see Equation (15.3.11)). We can differentiate the objective function with respect to the policy parameters θ, getting

$$\nabla_\theta J(\theta) = \int_\tau R(\tau) \nabla_\theta p(\tau;\theta) d\tau$$

which is intractable when the density p depends on θ. However, we can proceed as follows:

$$\begin{aligned} \nabla_\theta p(\tau;\theta) &= p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} \\ &= p(\tau;\theta) \nabla_\theta \log p(\tau;\theta) \end{aligned} \tag{17.1.6}$$

It acts like a change of probability measure. Plugging it back into the gradient of the expected reward, we get

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_\tau \Big(R(\tau) \nabla_\theta \log p(\tau;\theta) \Big) p(\tau;\theta) d\tau \\ &= E_{\tau \sim p(\tau;\theta)}[R(\tau) \nabla_\theta \log p(\tau;\theta)] \end{aligned}$$

There are several ways for estimating this expectation among which is the Monte Carlo sampling (see Appendix (26.2)).
We can write the density function for the trajectory τ as

$$p(\tau;\theta) = \prod_{t \geqslant 0} p(s_{t+1}|s_t, a_t) \pi_\theta(s_t, a_t)$$

Thus, taking the logarithm, we get

$$\log p(\tau;\theta) = \sum_{t \geqslant 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(s_t, a_t)$$

Differentiating with respect to θ, we get

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t \geqslant 0} \nabla_\theta \log \pi_\theta(s_t, a_t)$$

which does not depend on the transition probabilities $p(s_{t+1}|s_t, a_t)$. Replacing in the gradient, we get

$$\nabla_\theta J(\theta) = E_{\tau \sim p(\tau;\theta)}[R(\tau) \sum_{t \geqslant 0} \nabla_\theta \log \pi_\theta(s_t, a_t)] \tag{17.1.7}$$

As a result, when sampling a trajectory τ (sample $\{\tau_i\}$ from $\pi_\theta(s_t, a_t)$), we can estimate the performance $J(\theta)$ with the gradient estimator

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\nabla_\theta J(\theta) \approx R(\tau) \sum_{t \geqslant 0} \nabla_\theta \log \pi_\theta(s_t, a_t) \tag{17.1.8}$$

That is, when $R(\tau)$ is high, it push up the probabilities of the action seen, while when it is low the probabilities are pushed down.

***Remark* 17.1.4** *In the case where we consider all the sampling trajectories $\{\tau_i\}_{i=1}^N$, the gradient estimator becomes*

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \Big(\sum_{t \geqslant 0} \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t}) \Big) R(\tau_i)$$

where $R(\tau_i) = \sum_{t \geqslant 0} r(s_{t,i}, a_{t,i})$. The maximum likelihood is

$$\nabla_\theta J_{ML}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t \geqslant 0} \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t})$$

Policy gradient is on-policy. The REINFORCE algorithm is:

Algorithm 48 REINFORCE Algorithm

1: Sample $\{\tau_i\}$ from $\pi(s_t, a_t)$ (run the policy)
2: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \Big(\sum_{t \geqslant 0} \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t}) \Big) R(\tau_i)$
3: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

To simplify notation we write

$$\nabla_\theta \log \pi_\theta(\tau_i) = \sum_{t \geqslant 0} \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t})$$

so that the gradient estimator and the ML gradient simplify to

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_\theta \log \pi_\theta(\tau_i) R(\tau_i) \text{ and } \nabla_\theta J_{ML}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_\theta \log \pi_\theta(\tau_i)$$

The extra term makes the good trajectories more likely and the bad ones less likely. However, it is problematic in the case of good trajectories when $R(\tau)$ is negligible.
In this simplified notation the optimisation problem can be written as:

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[R(\tau)]$$

with gradient

$$\nabla_\theta J(\theta) = E_{\tau \sim p(\tau;\theta)}[\nabla_\theta \log \pi_\theta(\tau) R(\tau)]$$

In the MDP (see Definition (15.3.1)), the policy is a stochastic function representing a probability.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.1.4.1.1 Example As an example, we can consider Gaussian policies $\pi_\theta(s_t, a_t) = N(\mu_\theta(s_t), \Sigma_\theta)$ where $\mu_\theta(s_t) = E_{\pi_\theta}[a_t] = f_{network}(s_t)$ is the mean of the policy and $N(\cdot,\cdot)$ is the normal cumulative distribution function. We get

$$\log \pi_\theta(s_t, a_t) = -\frac{1}{2}\|f_{network}(s_t) - a_t\|^2 + C$$

where C is a constant. Its derivative is

$$\nabla_\theta \log \pi_\theta(s_t, a_t) = -\frac{1}{2}\Sigma_\theta^{-1}\big(f_{network}(s_t) - a_t\big)\frac{df}{d\theta}$$

***Remark* 17.1.5** *Note that this approach suffers from high variance since the credit assignment is hard.*

We must therefore find a way to reduce that variance.

17.1.4.2 Variance reduction

In the method presented in the previous section, the gradient is estimated by using Monte Carlo (MC) samples, which can have high variance. We discuss some solutions to improve the policy gradient by reducing the variance (see Appendix (26.2.2)). For simplicity of exposition we consider sampling a trajectory τ rather than looking at all trajectories at once.

17.1.4.2.1 Some solutions Some solutions of variance reduction are:

1. The policy at time $t^{'}$ does not affect the reward at time t when $t < t^{'}$. Thus, rather than using the total reward from inception $R(\tau) = \sum_{t\geqslant 0} r(s_t, a_t)$, we consider the reward to go $\sum_{t^{'}\geqslant t} r_{t^{'}}$ where $r_{t^{'}} = r(s_{t^{'}}, a_{t^{'}})$. That is, we push the probabilities up only by the cumulative future reward from that state. The gradient estimator becomes

$$\nabla_\theta J(\theta) \approx \sum_{t\geqslant 0}\big(\sum_{t^{'}\geqslant t} r_{t^{'}}\big)\nabla_\theta \log \pi_\theta(s_t, a_t)$$

   We let

$$\widehat{Q}_\pi(s_t, a_t) = \sum_{t^{'}\geqslant t} r_{t^{'}}$$

   be the reward to go if we take action a_t in state s_t. It is the Monte Carlo return from state s_t and action a_t. It estimates the expected reward if we take action $a_{t^{'}}$ in state $s_{t^{'}}$ (see Remark (15.3.3)). From Definition (15.3.6), the true expected reward to go, taking action a_t in state s_t, is

$$Q_\pi(s_t, a_t) = \sum_{t^{'}\geqslant t} E_{\pi_\theta}[r(s_{t^{'}}, a_{t^{'}})|s_t, a_t]$$

   so that $E_\pi[\widehat{Q}_\pi(s_t, a_t)] = Q_\pi(s_t, a_t)$.

2. Use discount factor γ to ignore delayed effects

$$\nabla_\theta J(\theta) \approx \sum_{t\geqslant 0}\big(\sum_{t^{'}\geqslant t} \gamma^{t^{'}-t} r_{t^{'}}\big)\nabla_\theta \log \pi_\theta(s_t, a_t)$$

   In that case, the discounted reward to go is $\widehat{Q}_\pi(s_t, a_t) = \sum_{t^{'}\geqslant t} \gamma^{t^{'}-t} r_{t^{'}}$.

However, the raw value of a trajectory is not necessarily meaningful. For instance, if rewards are all positive, we keep pushing up the probabilities of actions.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.1.4.2.2 Baseline function On the other hand, we are interested in whether a reward is better or worse than expected. Thus, one solution is to introduce a baseline function dependent on the state. That is, the gradient estimator is given by

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \sum_{t\geqslant 0}\Big(\sum_{t'\geqslant t}\gamma^{t'-t}r_{t'} - b(s_t)\Big)\nabla_\theta \log \pi_\theta(s_t,a_t) \\ &= \sum_{t\geqslant 0} A_t \nabla_\theta \log \pi_\theta(s_t,a_t)\end{aligned} \tag{17.1.9}$$

where $b(\cdot)$ is a baseline function. It corresponds to the gradient

$$\nabla_\theta J(\theta) = E_\pi[\sum_{t\geqslant 0}\nabla_\theta \log \pi_\theta(s_t,a_t)\big(R_t - b(s_t)\big)]$$

The baseline must be chosen carefully to reduce the variance. In general, it is a constant moving average of rewards experience so far from all the trajectories. That is, $b = \frac{1}{N}\sum_{i=1}^N R(\tau)$. This is because

$$\begin{aligned}E[\nabla_\theta \log \pi_\theta(\tau)b] &= \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)bd\tau \\ &= \int \nabla_\theta \pi_\theta(\tau)bd\tau \\ &= b\nabla_\theta \int \pi_\theta(\tau)d\tau = b\nabla_\theta 1 = 0\end{aligned}$$

since $\pi_\theta(\tau)$ is a density. As a result, the baseline is an unbiased expectation, but with some variance. See details of the policy gradient in Algorithm (49).

Algorithm 49 Policy Gradient Algorithm

Require: : training set S, number of training steps T
Initialise policy parameters θ
for $iteration = 1$ to N **do**
  Sample m trajectories under the current policy
  $\Delta\theta \leftarrow 0$
  for $i = 1$ to m **do**
    for $t = 1$ to T **do**
      $A_t = \sum_{t'\geqslant t}\gamma^{t'-t}r_{i,t} - b$
      $\Delta\theta \leftarrow \Delta\theta + A_t\nabla_\theta \log \pi_\theta(s_{i,t},a_{i,t})$
    end for
    $\theta \leftarrow \alpha\Delta\theta$
  end for
end for

17.1.5 The Actor-Critic algorithms

We saw in Section (17.1.4) that the policy gradients were on-policy methods directly maximising the cumulative future returns with respect to the policy. However, unless control variate, or other variance reduction methods, is applied, MC simulations suffer from high variance. Actor-critic methods (see Barto et al. [1983], Sutton [1984]) are based on simultaneous online parameter estimation of two structures to find an optimal policy:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1. Actor (control): it is an action selection policy mapping states to actions in a probabilistic manner.
2. Critic (prediction): it is a value function mapping states to expected cumulative future reward.

Marbach [1998] and Marbach et al. [1998] derived the gradient for the long-term expected reward per step. Sutton et al. [2000] extended this result to the start-state value function and provided simpler proofs. They proposed to use function approximation, such as actor-critic methods, which optimise the policy against a critic. The policy is represented by a function approximator, independent of the value function, and is updated with the gradient of expected reward with respect to the policy parameters. The gradient is estimated with the help of an approximate Q-value function or Advantage Function (the critic). The authors generalised previous approaches on actor-critic and policy-iteration architectures (see Barto et al. [1983], Sutton [1984]) and proved convergence. Other approaches on AC algorithms include Konda et al. [2000], Baxter et al. [2001], among others.

17.1.5.1 Policy gradient methods with approximator

We use the notation in Section (15.3.3).

17.1.5.1.1 Description The first result in Sutton et al. [2000] is the policy gradient theorem about the gradient of the performance metric with respect to the policy parameter:

Theorem 17.1.2 *For any MDP, in either the average-reward or start-state formulations, we have*

$$\frac{\partial \rho}{\partial \theta} = \sum_{s \in \mathcal{S}} d_\pi(s) \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a)}{\partial \theta} Q_\pi(s,a) \tag{17.1.10}$$

where ρ is the performance of the policy.

Note, in the continuous case, we have

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \int_{\mathcal{S}} \rho_\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(s,a) Q_\pi(s,a) dads \\
&= \int_{\mathcal{S}} \rho_\pi(s) \int_{\mathcal{A}} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) Q_\pi(s,a) dads \\
&= E_{s \sim \rho_\pi, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_\pi(s,a)]
\end{aligned}$$

Note, the discounted state distribution $d_\pi(s)$ is not differentiated with respect to θ. There is no effect of policy changes on the distribution of states, which simplifies the computation of the performance gradient. We can therefore approximate the gradient by sampling.
In general, Q_π is not known and must be estimated. If we use the return $R_t = \sum_{k=1}^{\infty} r_{t+k} - \rho(\pi)$ in the average-reward formulation (or $R_t = \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k}$ in the start-state formulation) as approximation for each $Q_\pi(s_t, a_t)$, we recover Equation (17.1.7) in the episodic REINFORCE algorithm. We can also approximate Q_π with a function approximator. Let $f_w : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be the approximation function to Q_π with parameter w. It is learned by following π and updating w by a rule such as

$$\Delta w_t \propto \frac{\partial}{\partial w} \left(\widehat{Q}_\pi(s_t, a_t) - f_w(s_t, a_t) \right)^2 \propto \left(\widehat{Q}_\pi(s_t, a_t) - f_w(s_t, a_t) \right) \frac{\partial f_w(s_t, a_t)}{\partial w}$$

where $\widehat{Q}_\pi(s_t, a_t)$ is an unbiased estimator of $Q_\pi(s_t, a_t)$ such as R_t. After that process has converged to a local optimum, we get

$$\sum_{s \in \mathcal{S}} d_\pi(s) \sum_{a \in \mathcal{A}} \pi(s,a) \big(Q_\pi(s,a) - f_w(s,a) \big) \frac{\partial f_w(s,a)}{\partial w} = 0$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Given the compatibility condition on f_w:

$$\begin{aligned}\frac{\partial f_w(s,a)}{\partial w} &= \frac{\partial \pi_\theta(s,a)}{\partial \theta}\frac{1}{\pi_\theta(s,a)} \\ &= \nabla_\theta \log \pi_\theta(s,a)\end{aligned}$$

and plugging it back in the above equation, we get

$$\sum_{s\in\mathcal{S}} d_\pi(s) \sum_{a\in\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \theta}\big(Q_\pi(s,a) - f_w(s,a)\big) = 0$$

***Remark* 17.1.6** *The error in $f_w(s,a)$ is orthogonal to the gradient of the policy parametrisation.*

As a result, the authors proved the following policy gradient with approximator theorem:

Theorem 17.1.3 *If f_w satisfies the above equations, then*

$$\frac{\partial \rho}{\partial \theta} = \sum_{s\in\mathcal{S}} d_\pi(s) \sum_{a\in\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \theta} f_w(s,a)$$

Assuming a policy that is a Gibbs distribution in a linear combination of features, where ϕ_{sa} is an l-dimensional feature vector characterising the pair (s,a), they obtained a natural parametrisation of f_w as

$$\begin{aligned}f_w(s,a) &= w^\top \nabla_\theta \log \pi_\theta(s,a) \\ &= w^\top\big(\phi_{sa} - \sum_b \pi(s,b)\phi_{sb}\big)\end{aligned} \tag{17.1.11}$$

***Remark* 17.1.7** *The approximation function f_w must be linear in the same features as the policy, except normalised to be mean zero for each state.*

They also stated that f_w could be thought as an approximation of the Advantage Function $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$ rather than of Q_π. Thus, the method was generalised to

$$\frac{\partial \rho}{\partial \theta} = \sum_s d_\pi(s) \sum_a \frac{\partial \pi(s,a)}{\partial \theta}\big(f_w(s,a) + v(s)\big)$$

where $v : \mathcal{S} \to \mathbb{R}$ is an arbitrary function which can affect the variance of the gradient estimators. From Equation (17.1.11), we can rewrite the gradient as

$$\begin{aligned}\frac{\partial \rho}{\partial \theta} &= \sum_{s\in\mathcal{S}} d_\pi(s) \sum_{a\in\mathcal{A}} \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\big(f_w(s,a) + v(s)\big) \\ &= E_{s\sim\rho_\pi, a\sim\pi_\theta}\big[\nabla_\theta \log \pi_\theta(s,a)\big(f_w(s,a) + v(s)\big)\big]\end{aligned} \tag{17.1.12}$$

This is similar to the use of the baseline function in Section (17.1.4.2). Thus, it should be set to the best estimator of V_π.
Then the authors proved convergence. Given $\{\alpha_k\}_{k=0}^\infty$ a step-wise sequence such that $\lim_{k\to\infty} \alpha_k = 0$ and $\sum_k \alpha_k = \infty$, for any MDP with bounded rewards, the sequence $\{\rho(\pi_k)\}_{k=0}^\infty$, defined by $\pi_k = \pi(\cdot,\cdot,\theta_k)$ and w_k satisfying the conditions

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} w_k &= w \text{ such that } \sum_{s\in\mathcal{S}} d_{\pi_k}(s) \sum_{a\in\mathcal{A}} \pi_k(s,a)\big(Q_{\pi_k}(s,a) - f_w(s,a)\big)\frac{\partial f_w(s,a)}{\partial w} = 0 \\ \theta_{k+1} &= \theta_k + \alpha_k \frac{\partial \rho(\pi_k)}{\partial \theta} \end{aligned} \tag{17.1.13}$$

with

$$\begin{aligned} \frac{\partial \rho(\pi_k)}{\partial \theta} &= \sum_{s\in\mathcal{S}} d_\pi(s) \sum_{a\in\mathcal{A}} \frac{\partial \pi_k(s,a)}{\partial \theta}\big(f_{w_k}(s,a) + v(s)\big) \\ &= \sum_{s\in\mathcal{S}} d_\pi(s) \sum_{a\in\mathcal{A}} \pi_k(s,a)\nabla_\theta \log \pi_k(s,a)\big(f_{w_k}(s,a) + v(s)\big) \end{aligned}$$

The step-wise sequence converges such that $\lim_{k\to\infty} \frac{\partial \rho(\pi_k)}{\partial \theta} = 0$.

17.1.5.1.2 Variance reduction Bhatnagar et al. [2007] extended the results of Sutton et al. [2000]. Taking the gradient of the average reward for parametrised policies in Equation (17.1.10) and replacing $Q_\pi(s,a)$ with a baseline function $b(s)$ we get

$$\begin{aligned} \frac{\partial \rho}{\partial \theta} &= \sum_s d_\pi(s) \sum_a \nabla_\theta \pi(s,a) b(s) \\ &= \sum_s d_\pi(s) b(s) \nabla_\theta \big(\sum_a \pi(s,a)\big) \\ &= \sum_s d_\pi(s) b(s) \nabla_\theta (1) = 0 \end{aligned}$$

Thus, for any baseline $b(s)$, the gradient of the average reward can be written as

$$\nabla_\theta \rho(\pi) = \sum_{s\in\mathcal{S}} d_\pi(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi(s,a)\big(Q_\pi(s,a) \pm b(s)\big) \tag{17.1.14}$$

where the baseline should be chosen in such a way as to minimise the variance of the gradient estimates.
We saw in the previous section that when we approximate the true action-value function $Q_\pi(s,a)$ with the approximator $Q_{\pi,\phi}(s,a)$, we introduce a bias. In order to reduce that bias, the approximator $Q_{\pi,\phi}(s,a)$ must be compatible. That is:

1. From Equation (17.1.11) and Remark (17.1.7), the function approximator $Q_{\pi,\phi}(s,a)$ must be linear in features of the stochastic policy. In our case, the features are $\nabla_\theta \log \pi_\theta(s,a)$.

2. From Remark (17.1.6), the parameters ϕ must be chosen to minimise the mean-squared error

$$\epsilon_\pi(\phi) = \sum_s d_\pi(s) \sum_a \pi(s,a)\Big(Q_\pi(s,a) - Q_{\pi,\phi}(s,a)\Big)^2$$

   to reduce the bias. Thus, we can replace $Q_\pi(s,a)$ with $Q_{\pi,\phi*}(s,a)$.

As a result of [1], we work with the linear approximation

$$Q_{\pi,\phi}(s,a) = \phi^\top \psi(s,a)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where

$$\psi(s,a) = \nabla_\theta \log \pi_\theta(s,a)$$

We can also consider the mean-squared error

$$\epsilon_\pi(\phi) = \sum_s d_\pi(s) \sum_a \pi(s,a) \Big(Q_\pi(s,a) - \phi^\top \psi(s,a) - b(s) \Big)^2$$

with an arbitrary baseline $b(s)$. We let $\phi^* = \arg\min_\phi \epsilon_\pi(\phi)$ be the optimal parameter. Then, ϕ^* does not depend on the baseline $b(s)$. Thus, both mean squared error problems have the same solution. The authors showed that the mean squared error $\epsilon_\pi(\phi^*)$ was further minimised when $b(s) = V_\pi(s)$. That is, when estimating the Q-value function, the variance is minimised if the baseline function is equal to the value function.

Lemma 17.1.1 *For any given policy π, the minimum variance baseline $b^*(s)$ in the action-value function estimator corresponds to the value function $V_\pi(s)$.*

As a result, $(\phi^*)^\top \psi(s,a)$ is a least-squared optimal parametric representation for the Advantage Function $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$ as well as for the Q-value function $Q_\pi(s,a)$. However, since

$$E_{a\sim\pi}[\phi^\top \psi(s,a)] = \sum_{a\in\mathcal{A}} \pi(s,a)\phi^\top \psi(s,a) = 0 \,, \forall s \in \mathcal{S}$$

one should rather think of $\phi^\top \psi(s,a)$ as an approximation to the Advantage Function.

17.1.5.1.3 The temporal-difference method Usually, in case [2], practitioners estimate the value function with a temporal-difference (TD) learning (see Section (17.1.2.4)). We briefly present the approach proposed by Bhatnagar et al. [2007] and Degris et al. [2012b].
We let the TD error for the average reward be defined by

$$\sigma_t = r_{t+1} - \hat{\rho}_{t+1} + \hat{V}(s_{t+1}) - \hat{V}(s_t)$$

where $\hat{V}$ and $\hat{\rho}$ are consistent estimates of the value function and average reward, respectively. Thus, they satisfy $E_\pi[\hat{V}(s_t)|\, s_t] = V_\pi(s)$ and $E_\pi[\hat{\rho}_{t+1}|\, s_t] = \rho(\pi)$, for any $t \geqslant 0$. The authors showed that σ_t was a consistent estimate of the Advantage Function A_π.

Lemma 17.1.2 *Given policy π, we have $E_\pi[\sigma_t|\, s_t, a_t] = A_\pi(s_t, a_t)$.*

Setting $b(s) = V_\pi(s)$ in Equation (17.1.14), and using the compatibility condition [1], we get

$$\begin{aligned}
\nabla_\theta \rho(\pi) &= \sum_{s\in\mathcal{S}} d_\pi(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi(s,a) A_\pi(s,a) \\
&= \sum_{s\in\mathcal{S}} d_\pi(s) \sum_{a\in\mathcal{A}} \pi(s,a) \nabla_\theta \log \pi_\theta(s,a) A_\pi(s,a) \\
&= \sum_{s\in\mathcal{S}} d_\pi(s) \sum_{a\in\mathcal{A}} \pi(s,a) \psi(s,a) A_\pi(s,a) \\
&= E_{s\sim\rho_\pi, a\sim\pi_\theta}[\psi(s,a) A_\pi(s,a)]
\end{aligned}$$

Thus, given Lemma (17.1.2), one can deduce that $\widehat{\nabla_\theta \rho(\pi_\theta)} = \sigma_t \psi(s_t, a_t)$ is a consistent estimate of $\nabla_\theta \rho(\pi_\theta)$. However, it is not that simple to compute the estimate $\hat{V}$. It can be approximated. Suppose $f(s)$ is a feature vector for state

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

s. We can approximate $V_\pi(s)$ with $v^\top f(s)$, where v is a parameter vector that can be tuned with a TD algorithm for a fixed policy π. The authors used

$$\sigma_t = r_{t+1} - \widehat{\rho}_{t+1} + v_t^\top f(s_{t+1}) - v_t^\top f(s_t)$$

as an estimate for the TD error, where v_t is the parameter vector at time t.
The authors defined

$$\overline{V}_\pi(s) = \sum_{a\in\mathcal{A}} \pi(s,a)\big[R(s,a) - \rho(\pi) + \sum_{s'\in\mathcal{S}} p(s' \mid s,a) v_\pi^\top f(s')\big]$$

where $v_\pi^\top f(s')$ is an estimate of $V_\pi(s')$ in the limit $t \to \infty$. They also defined

$$\sigma_{t,\pi} = r_{t+1} - \widehat{\rho}_{t+1} + v_\pi^\top f(s_{t+1}) - v_\pi^\top f(s_t)$$

where $\sigma_{t,\pi}$ is a stationary estimate of the TD error. This led to the following lemma:

Lemma 17.1.3 $E_\pi[\sigma_{t,\pi}\psi(s_t,a_t)] = \nabla_\theta \rho(\pi) + \sum_{s\in\mathcal{S}} d_\pi(s)\big[\nabla_\theta \overline{V}_\pi(s) - \nabla_\theta v_\pi^\top f(s)\big]$

The quantity $\sum_{s\in\mathcal{S}} d_\pi(s)\big[\nabla_\theta \overline{V}_\pi(s) - \nabla_\theta v_\pi^\top f(s)\big]$ is the error (or bias) in the estimate of the gradient of average reward resulting from function approximation.
The authors introduced four AC algorithms using the TD error to estimate the Q-value function. We display in Algorithm (50) the one estimating the regular gradient.

Algorithm 50 Incremental Actor-Critic Algorithm

Require: : Randomised parametrised policy $\pi(\cdot,\cdot;\theta)$ and the value function feature vector $f(s)$.
Initialise policy parameters $\theta = \theta_0$, value function vector $v = v_0$.
Initialise step sizes $\alpha = \alpha_0$, $\beta = \beta_0$, $\xi = \xi_0$, and state s_0.
for $t = 0$ to T **do**
  Draw action $a_t \sim \pi(s_t, a_t; \theta_t)$
  Observe next state $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$
  Observe reward r_{t+1}
  Average reward update: $\widehat{\rho}_{t+1} = (1-\xi_t)\widehat{\rho}_t + \xi_t r_{t+1}$
  TD error: $\sigma_t = r_{t+1} - \widehat{\rho}_{t+1} + v_t^\top f(s_{t+1}) - v_t^\top f(s_t)$
  Critic update: $v_{t+1} = v_t + \alpha_t \sigma_t f(s_t)$
  Actor update: $\theta_{t+1} = \theta_t + \beta_t \sigma_t \psi(s_t, a_t)$
end for
return Policy and value function parameters θ, v.

The authors defined convergence in a weak sense.

Theorem 17.1.4 *For the parameters in Algorithm (50), we have* $(\widehat{\rho}_t, v_t, \theta_t) \to \{(\rho(\theta^*), v_{\theta^*}, \theta^*);\ \theta^* \in Z\}$ *as* $t \to \infty$ *with probability one, where the set* Z *corresponds to the set of local maxima of a performance function with gradient* $E_\pi[\sigma_{t,\pi}\psi(s_t,a_t)]$.

17.1.5.2 Description

The actor-critic methods are based on the policy gradient theorem (see Theorem (17.1.2)). Various authors derived policy gradient algorithms such as Peters et al. [2005], Bhatnagar et al. [2007], Degris et al. [2012], among others.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.1.5.2.1 Overview Recall from variance reduction in Section (17.1.4.2), we want to push up the probability of an action from a state if this action is better than the expected value of what we should get from that state. The latter is the Q-function and value function described in Section (15.3.3). Recall, the true expected reward to go $Q(s_t, a_t)$ from taking action a_t in state s_t is defined above and the total reward from s_t (see Equation (15.3.19)) is given by the value function

$$V_\pi(s_t) = E_{a_t \sim \pi_\theta(s_t, a_t)}[Q_\pi(s_t, a_t)]$$

Formally, given the Advantage Function in Equation (17.1.1), we want to favour an action a_t in a state s_t if the action

$$\begin{aligned} A_\pi(s_t, a_t) &= Q_\pi(s_t, a_t) - V_\pi(s_t) \\ &= Q_\pi(s_t, a_t) - E_{a_t \sim \pi_\theta(s_t, a_t)}[Q_\pi(s_t, a_t)] \end{aligned}$$

is large. That is, the Advantage Function $A_\pi(s_t, a_t)$ is a centered Q-value function around it mean, the value function V_π. This is a standard variance reduction technique (see Appendix (26.2.2)) which reduces the variance of the gradient. Q_π measures the performance of each action from a given state, assuming it follows policy π afterwards, and A_π provides a measure of how each action compares to the average performance at that state. Further, we want to discard the action if that difference is small. Thus, accounting for this, we can modify the gradient estimator accordingly, getting

$$\begin{aligned} \nabla_\theta J(\theta) &\approx \sum_{t \geqslant 0} \Big(Q_{\pi_\theta}(s_t, a_t) - V_{\pi_\theta}(s_t) \Big) \nabla_\theta \log \pi_\theta(s_t, a_t) \\ &= \sum_{t \geqslant 0} A_\pi(s_t, a_t) \nabla_\theta \log \pi_\theta(s_t, a_t) \end{aligned} \tag{17.1.15}$$

and we recover the generalised gradient in Equation (17.1.12) (see also Equation (17.1.14)).
The better the Advantage Function $A_\pi(s_t, a_t)$, the lower the variance. This is to compare with the baseline function in Equation (17.1.9), which is unbiased, but has high variance. Note, we have $b(s_t) \approx V_{\pi_\theta}(s_t)$ as in Lemma (17.1.1). Recall from Equation (15.3.20), the Q-value function seen at time t is

$$Q_{\pi_\theta}(s_t, a_t) = r(s_t, a_t) + \gamma E_{s_{t+1} \sim p(s_{t+1}|(s_t, a_t))}[V_\pi(s_{t+1})]$$

where $\gamma \in [0, 1]$ is the discount factor. Putting terms together, the Advantage Function becomes

$$A_\pi(s_t, a_t) = r(s_t, a_t) + \gamma E_{s_{t+1} \sim p(s_{t+1}|(s_t, a_t))}[V_\pi(s_{t+1})] - V_\pi(s_t)$$

We now need to fit a model to estimate the functions Q_π, V_π, or A_π and define some actor-critic algorithms. As discussed above, the actor-critic is made of two components:

- An actor adjusts the parameters θ of the stochastic policy $\pi_\theta(s, a)$ via stochastic gradient ascent.
- A critic estimates the action-value function $Q_{\pi,\phi}(s, a)$, with parameters ϕ, that approximate the unknown true action-value function $Q_\pi(s, a)$. That is, $Q_{\pi,\phi}(s, a) \approx Q_\pi(s, a)$ by using an appropriate policy evaluation algorithm.

We discuss off-policy variant of actor-critic algorithms in Section (17.1.7).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.1.5.2.2 The value function We now need to perform policy evaluation. A simple approximation to the value function $V_\pi(s_t)$ (see Definition (15.3.4)) is

$$V_\pi(s_t) \approx \sum_{t' \geqslant t} r(s_{t'}, a_{t'})$$

which we denote $\widehat{V}_\pi(s_t)$. Replacing in the above expectation, the Advantage Function is approximated as

$$A_{\pi_\theta}(s_t, a_t) \approx r(s_t, a_t) + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)$$

This is similar to the temporal difference (TD) method described in Section (17.1.5.1), but with a different estimate for the value function.
A better approximation is to apply Monte Carlo simulation to approximate the value function as follows:

$$V_\pi(s_t) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t' \geqslant t} r(s_{i,t'}, a_{i,t'})$$

Alternatively, we can use a neural network with parameters ϕ to fit $\widehat{V}_\pi(s)$ and get the supervised regression

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \|\widehat{V}_{\pi,\phi}(s_i) - y_i\|^2$$

where y_i is the target value. The updating rule for ϕ is

$$\Delta\phi_t \propto \nabla_\phi \big(V_{\pi,\phi}(s_t) - y_t\big)^2 \propto \big(V_{\pi,\phi}(s_t) - y_t\big) \nabla_\phi V_{\pi,\phi}(s_t)$$

In general, the target value is some kind of unbiased estimator of $V_\pi(s_t)$ (see above). We obtain the training data $\{(s_{i,t}, y_{i,t}); i = 1, ..., N\}$ where the target is $y_{i,t} = \sum_{t' \geqslant t} r(s_{i,t'}, a_{i,t'})$.
Another solution is to use the previously fitted value function to approximate the next value function (see Section (17.1.3.4)). Given Equation (15.3.13), we get

$$\begin{aligned} y_{i,t} = \sum_{t' \geqslant t} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_{i,t}] &= r(s_{i,t}, a_{i,t}) + \gamma E_{s_{i,t+1} \sim p(s_{i,t+1}|(s_{i,t}))}[V_\pi(s_{i,t+1})] \\ &\approx r(s_{i,t}, a_{i,t}) + \gamma V_\pi(s_{i,t+1}) \\ &\approx r(s_{i,t}, a_{i,t}) + \gamma \widehat{V}_{\pi,\phi}(s_{i,t+1}) \end{aligned}$$

so that the target in the training data becomes

$$y_{i,t} = r(s_{i,t}, a_{i,t}) + \gamma \widehat{V}_{\pi,\phi}(s_{i,t+1})$$

At every step t, the target is the previously fitted value function incremented with the reward at that time. Thus, we are performing a bootstrapping to estimate the model parameters. Putting terms together, the Advantage Function is approximated as

$$A_{\pi_\theta,C}(s_{i,t}, a_{i,t}) \approx r(s_{i,t}, a_{i,t}) + \gamma \widehat{V}_{\pi,\phi}(s_{i,t+1}) - \widehat{V}_{\pi,\phi}(s_{i,t}) \tag{17.1.16}$$

17.1.5.2.3 The Q-value function As discussed above, the actor-critic methods consist in:

1. a policy evaluation step, which learns to fit a critic $Q_{\pi,\phi}$ for the current policy π_θ, and
2. an actor performing a policy improvement step, which greedily optimises the policy π_θ against the critic estimate $Q_{\pi,\phi}$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Similarly to the above approach taken with the value function, we can use a neural network with parameters ϕ to fit $Q_\pi(s,a)$ and get the supervised regression

$$\mathcal{L}(\phi) = \frac{1}{2}\sum_i \|Q_{\pi,\phi}(s_i,a_i) - y_i\|^2$$

where y_i is the target value. The updating rule for ϕ is

$$\Delta\phi_t \propto \nabla_\phi\big(Q_{\pi,\phi}(s_t,a_t) - y_t\big)^2 \propto \big(Q_{\pi,\phi}(s_t,a_t) - y_t\big)\nabla_\phi Q_{\pi,\phi}(s_t,a_t)$$

In general, the target value is some kind of unbiased estimator of $Q_\pi(s_t,a_t)$. Given Equation (15.3.18), we get

$$\begin{aligned} y_{i,t} = \sum_{t'\geqslant t} E_{\pi_\theta}[r(s_{t'},a_{t'})|(s_{i,t},a_{i,t})] &= r(s_{i,t},a_{i,t}) + \gamma E_{s_{i,t+1}\sim p(s_{i,t+1}|(s_{i,t},a_{i,t}))}[Q_\pi(s_{i,t+1},a_{i,t+1})] \\ &\approx r(s_{i,t},a_{i,t}) + \gamma Q_\pi(s_{i,t+1},a_{i,t+1}) \\ &\approx r(s_{i,t},a_{i,t}) + \gamma Q_{\pi,\phi}(s_{i,t+1},a_{i,t+1}) \end{aligned}$$

so that the target in the training data becomes

$$y_{i,t} = r(s_{i,t},a_{i,t}) + \gamma Q_{\pi,\phi}(s_{i,t+1},a_{i,t+1})$$

If we do not know the action a_{t+1}, we can approximate it with its mean $\mu_\theta(s_{t+1}) = E_{\pi_\theta}[a_{t+1}]$.

17.1.5.2.4 The main steps We therefore need to perform the following steps:

1. Estimate $A_{\pi_\theta}(s_t,a_t)$ for current policy π.
2. Use $A_{\pi_\theta}(s_t,a_t)$ to get improved policy $\pi^{'}$.
3. Repeat step [1].

We will show in Section (17.1.7.1) that this is similar to the policy iteration algorithm:

Algorithm 51 Policy Iteration Algorithm

1: Estimate $A_{\pi_\theta}(s,a)$.
2: Set $\pi \leftarrow \pi^{'}$.
3: Repeat step [1].

17.1.5.3 The algorithm

We can now introduce the Actor-Critic algorithm: If we do not know the values Q and V, can we learn them? We saw in Section (17.1.3) that we could do it with the Q-learning algorithms. We can also combine Policy Gradients and Q-learning by training both an actor (the policy) and a critic (the Q-function).
The actor is the policy π deciding which action to take, and the critic tells the actor (via the value function or Q-value function) how good its action was and how it should be adjusted. Note, we can incorporate the experience replay and alleviate the task of the critic. We let the Advantage Function A_π define how much an action a is better than the expected action according to the policy π (see Equation (17.1.1)). It is given by

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In the case of the actor-critic algorithm, the Advantage Function is given in Equation (17.1.16). The batch actor-critic algorithm is given in Algorithm (52).

Algorithm 52 Batch Actor-Critic Algorithm

Require: : training set S, number of training steps T

Initialise policy parameters θ, critic parameters ϕ
Sample $\{s_i, a_i\}$ from $\pi_\theta(s, a)$.
Fit $\widehat{V}_{\pi,\phi}(s)$ to sampled reward sums.
Evaluate $\widehat{A}_\pi(s_i, a_i) = r(s_i, a_i) + \gamma \widehat{V}_{\pi,\phi}(s_i^{'}) - \widehat{V}_{\pi,\phi}(s_i)$.
Compute $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(a_i, s_i)\widehat{A}_\pi(s_i, a_i)$.
Update $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$.

See details of the Actor-Critic in Algorithm (53).

Algorithm 53 Actor-Critic Algorithm

Require: : training set S, number of training steps T

Initialise policy parameters θ, critic parameters ϕ
Initialise action-value function Q with random weights
for $iteration = 1$ to N **do**
  Sample m trajectories under the current policy
  $\Delta\theta \leftarrow 0$
  for $i = 1$ to m **do**
    for $t = 1$ to T **do**
      $A_t = r(s_{i,t}, a_{i,t}) + \gamma \widehat{V}_{\pi,\phi}(s_{i,t+1}) - \widehat{V}_{\pi,\phi}(s_{i,t})$
      $\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t})$
    end for
  end for
  $\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi \|A_{i,t}\|^2$
  $\theta \leftarrow \alpha\Delta\theta$
  $\phi \leftarrow \beta\Delta\phi$
end for

We can also write an online actor-critic algorithm, see Algorithm (54).

Algorithm 54 Online Actor-Critic Algorithm

Require: : training set S, number of training steps T

Initialise policy parameters θ, critic parameters ϕ
Take action $a \sim \pi_\theta(s, a)$, get $(s, a, s^{'}, r)$
Update $\widehat{V}_{\pi,\phi}$ using target $r + \gamma \widehat{V}_{\pi,\phi}(s^{'})$
Evaluate $\widehat{A}_\pi(s, a) = r(s, a) + \gamma \widehat{V}_{\pi,\phi}(s^{'}) - \widehat{V}_{\pi,\phi}(s)$
$\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(s, a)\widehat{A}_\pi(s, a)$
Update $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.1.6 Gradient base actor-critic

17.1.6.1 State-dependent baselines

We can combine the policy gradient (see Section (17.1.4)) with the actor-critic to get an unbiased expectation with lower variance. In that case, combining the gradient in Equation (17.1.9) with that in Equation (17.1.15), we get

$$\nabla_\theta J(\theta) \approx \sum_{t\geqslant 0}\Big(\sum_{t'\geqslant t}\gamma^{t'-t}r_{t'} - \widehat{V}_{\pi,\phi}(s_t)\Big)\nabla_\theta \log \pi_\theta(s_t,a_t) \tag{17.1.17}$$

where

$$A_{\pi_\theta,MC}(s_t,a_t) \approx \sum_{t'\geqslant t}\gamma^{t'-t}r_{t'} - \widehat{V}_{\pi,\phi}(s_t)$$

In that setting, the critics become state-dependent baselines (see Section (17.1.4.2)). See details of the implementation in Algorithm (55).
Assuming n-step returns, we can also combine $A_{\pi_\theta,MC}$ with $A_{\pi_\theta,C}$ to reduce the variance, getting

$$A_{\pi_\theta,n}(s_t,a_t) \approx \sum_{t'=t}^{t+n}\gamma^{t'-t}r_{t'} - \widehat{V}_{\pi,\phi}(s_t) + \gamma^n\widehat{V}_{\pi,\phi}(s_{t+n}) \tag{17.1.18}$$

where $n > 1$.

Algorithm 55 Gradient Base Actor-Critic Algorithm

Require: : training set S, number of training steps T
Initialise policy parameters θ, critic parameters ϕ
Initialise action-value function Q with random weights
for $iteration = 1$ to N **do**
  Sample m trajectories under the current policy
  $\Delta\theta \leftarrow 0$
  for $i = 1$ to m **do**
    for $t = 1$ to T **do**
      $A_t = \sum_{t'\geqslant t}\gamma^{t'-t}r_{i,t} - \widehat{V}_{\pi,\phi}(s_{i,t})$
      $\Delta\theta \leftarrow \Delta\theta + A_t\nabla_\theta \log \pi_\theta(s_{i,t},a_{i,t})$
    end for
  end for
  $\Delta\phi \leftarrow \sum_i\sum_t \nabla_\phi\|A_{i,t}\|^2$
  $\theta \leftarrow \alpha\Delta\theta$
  $\phi \leftarrow \beta\Delta\phi$
end for

17.1.6.2 Continuous action methods

17.1.6.2.1 The deterministic policy gradient Silver et al. [2014] proposed the deterministic policy gradient (DPG) for reinforcement learning with continuous action spaces. Rather than performing a global maximisation at every step, the authors moved the policy in the direction of the gradient of Q (see Remark (17.1.6)). For each visited state s, the policy parameters θ_{k+1} are updated in proportion to the gradient $\nabla_\theta Q_\pi(s,\mu_\theta(s))$ with deterministic policy μ_θ. Recall, we can express the maximum as

$$\max_a Q_\theta(s,a) = Q_\theta\big(s, \arg\max_a Q_\theta(s,a)\big)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Here, the deterministic policy is the greedy policy $\mu_{k+1}(s) = \arg\max_a Q_{\mu_k}(s,a)$ (see Definition (15.3.7)). This is to compare with the stochastic policy $\pi_k = \pi(s,a,\theta_k)$ in Section (17.1.5.1).
The deterministic policy is given by $\pi_\theta(s_t,a_t) = \delta_{a_t=\mu_\theta(s_t)}$, where $\mu_\theta(s_t) = E_{\pi_\theta}[a_t]$ is the mean of the policy, and β is an arbitrary exploration distribution for a_t. It is a limit to the stochastic policies $\pi_{\mu_\theta,\beta}$ when $\beta = 0$. The authors derived an off-policy actor-critic algorithm that estimates the Q-function with a differentiable function approximator, and update the policy parameters in the direction of the approximate Q-gradient.
Given the above deterministic policy, the authors considered the performance objective:

$$\begin{aligned} J(\theta) &= \int_S \rho_\mu(s) R(s_t,\mu_\theta(s_t))ds \\ &= E_{s_t\sim\rho_\mu}[R(s_t,\mu_\theta(s_t))] \end{aligned}$$

In the case of an off-policy, with stochastic policy π_θ, the authors considered the modified objective given by the value function of the target policy, averaged over the state distribution of the policy:

$$\begin{aligned} J(\theta) &= \int_S \rho_\pi(s) V_\pi(s)ds \\ &= E_{s_t\sim p(s_t;\beta)}[Q_\pi(s_t,\mu_\theta(s_t))] \end{aligned}$$

where $p(s_t;\beta)$ is the state distribution, and the policy parameters are given by

$$\theta = \arg\max_\theta E_{s_t\sim p(s_t;\beta)}[Q_\pi(s_t,\mu_\theta(s_t))]$$

When using neural networks, full optimisation is expensive, so that stochastic gradient optimisation is used. Here, the policy gradient only integrates over the state space. Using the chain rule, the authors proposed the deterministic policy gradient theorem, which states that the gradient in the policy improvement phase is given by

$$\begin{aligned} \nabla_\theta J(\theta) &\approx E_{s_t\sim p(s_t;\beta)}[\nabla_\theta Q_\pi(s_t,\mu_\theta(s_t))] \\ &= E_{s_t\sim p(s_t;\beta)}[\nabla_a Q_\pi(s_t,a)\big|_{a=\mu_\theta(s_t)} \nabla_\theta \mu_\theta(s_t)] \end{aligned}$$

where $\nabla_\theta \mu_\theta(s_t)$ is a Jacobian matrix. The policy improvement is given by

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta)$$

The authors used that theorem to derive both on-policy and off-policy actor-critic algorithms. They followed the temporal difference (TD) method described in Section (17.1.5.1). The true action-value function Q_π is replaced with the function approximator $Q_{\pi,\phi}$ and a critic estimates $Q_{\pi,\phi}(s,a) \approx Q_\pi(s,a)$ using a policy evaluation algorithm. The latter is as follows:

- The on-policy: The authors considered the Sarsa critic (see Section (17.1.2.4)):

$$\begin{aligned} \sigma_t &= r_t + \gamma Q_{\pi,\phi}(s_{t+1},a_{t+1}) - Q_{\pi,\phi}(s_t,a_t) \\ \phi_{t+1} &= \phi_t + \alpha_\phi \sigma_t \nabla_\phi Q_{\pi,\phi}(s_t,a_t) \\ \theta_{t+1} &= \theta_t + \alpha_\theta \nabla_a Q_{\pi,\phi}(s_t,a_t)\big|_{a=\mu_\theta(s_t)} \nabla_\theta \mu_\theta(s_t) \end{aligned}$$

- The off-policy: The trajectories are generated by an arbitrary stochastic policy $\pi(s,a)$. The critic uses Q-learning (see Section (17.1.3)) updates to estimate the action-value function:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}\sigma_t &= r_t + \gamma Q_{\pi,\phi}(s_{t+1}, \mu_\theta(s_{t+1})) - Q_{\pi,\phi}(s_t, a_t)\\ \phi_{t+1} &= \phi_t + \alpha_\phi \sigma_t \nabla_\phi Q_{\pi,\phi}(s_t, a_t)\\ \theta_{t+1} &= \theta_t + \alpha_\theta \nabla_a Q_{\pi,\phi}(s_t, a_t)\big|_{a=\mu_\theta(s_t)} \nabla_\theta \mu_\theta(s_t)\end{aligned}$$

Since the policy is deterministic, there is no integral over the actions (see Equation (17.1.13)), and no need to perform importance sampling in the actor.

While it is not prone to high variance and is trainable on off-policy data, it produces a biased policy gradient estimator, making it difficult to analyse convergence and stability properties. This is to oppose to the baseline gradient in Equation (17.1.9), which provides an almost unbiased gradient, but with high variance.

17.1.6.2.2 Extensions Accounting for this result, Gu et al. [2017] proposed to use the deterministic biased estimator as a control variate for the MC policy gradient estimator, getting action-dependent baselines. It is the combination of an on-policy with an off-policy, which brings stability and efficiency. The off-policy is given by

$$\nabla_\theta J_{off}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t\geqslant 0} \nabla_\theta E_{a\sim\pi_\theta(s_{i,t},a_t)}[Q_{\pi,\phi}(s_{i,t}, \mu_\theta(s_t))]$$

which is given as above. The on-policy is given by

$$\nabla_\theta J_{on}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t\geqslant 0}\Big(\widehat{Q}_i(s_t, a_t) - \overline{Q}_{\pi,\phi}(s_{i,t}, a_{i,t})\Big)\nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t})$$

where $\overline{Q}_{\pi,\phi}(s_t, a_t)$ is a first order Taylor expansion around $a_t = \mu_\theta(s_t)$. In that case, the gradient estimator is given by

$$\nabla_\theta J(\theta) \approx \nabla_\theta J_{on}(\theta) + \nabla_\theta J_{off}(\theta) \tag{17.1.19}$$

This result holds provided that the second term can be evaluated.
Recall, to control the bias and reduce the variance, it was also suggested to combine $A_{\pi_\theta,C}$ with $A_{\pi_\theta,MC}$ (see Equation (17.1.18)). Rather than choosing a single n, Schulman et al. [2016] suggested to get a weighted combination of n-step returns called the generalised advantage estimation (GAE), given by

$$\widehat{A}_{\pi,GAE}(s_t, a_t) = \sum_{n=1}^{\infty} \omega_n \widehat{A}_{\pi_\theta,n}(s_t, a_t)$$

Letting $\omega_n \propto \lambda^{n-1}$, we get

$$\widehat{A}_{\pi,GAE}(s_t, a_t) = r(s_t, a_t) + \gamma\Big[(1-\lambda)\widehat{V}_{\pi,\phi}(s_{t+1}) + \lambda r(s_{t+1}, a_{t+1}) + \gamma\big((1-\lambda)V_{\pi,\phi}(s_{t+2}) + \lambda r(s_{t+2}, a_{t+2}) + \cdots\Big]$$

which we rewrite more compactly as

$$\widehat{A}_{\pi,GAE}(s_t, a_t) = \sum_{t'\geqslant t}(\gamma\lambda)^{t'-t}\delta_{t'}$$

where

$$\delta_{t'} = r(s_{t'}, a_{t'}) + \gamma\widehat{V}_{\pi,\phi}(s_{t'+1}) - \widehat{V}_{\pi,\phi}(s_{t'})$$

and $(\gamma\lambda)$ acts as a discount rate.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.1.7 Policy gradient improvement

We have seen in Section (17.1.4) that policy gradient were on-policy. We are now going to derive off-policy variant by using importance sampling.

17.1.7.1 Policy iteration

We present the policy iteration introduced by Kakade et al. [2002]. We start from the objective function in Equation (17.1.5) and use the decomposition in Equation (15.3.13). Recall,

$$V_{\pi_\theta}(s_t) = \sum_{t^{'} \geqslant t} E_{\pi_\theta}[\gamma^{t^{'}} r_{t^{'}} | s_t] \, , r_{t^{'}} = r(s_{t^{'}}, a_{t^{'}})$$

and assuming the chain to be homogeneous (see Definition (24.6.2)), we get the objective

$$J(\theta) = E_{s_1 \sim p(s_1)}[V_{\pi_\theta}(s_1)]$$

Further, $E_{\tau \sim p(\tau;\theta^{'})}[\cdot]$ indicates that actions are sampled from $\pi_{\theta^{'}}$ to generate the trajectory $\tau = (s_0, a_0, s_1, a_1, ...)$. We are interested in computing the difference in policy performance $J(\theta^{'}) - J(\theta)$. The authors showed that it could be decomposed as a sum of per-timestep advantages. Starting at $t = 0$, we get

$$\begin{aligned}
J(\theta^{'}) - J(\theta) &= J(\theta^{'}) - E_{s_0 \sim p(s_0)}[V_{\pi_\theta}(s_0)] \\
&= J(\theta^{'}) - E_{\tau \sim p(\tau;\theta^{'})}[V_{\pi_\theta}(s_0)] \\
&= J(\theta^{'}) - E_{\tau \sim p(\tau;\theta^{'})}[\sum_{t=0}^{\infty} \gamma^t V_{\pi_\theta}(s_t) - \sum_{t=1}^{\infty} \gamma^t V_{\pi_\theta}(s_t)] \\
&= J(\theta^{'}) + E_{\tau \sim p(\tau;\theta^{'})}[\sum_{t=0}^{\infty} \gamma^t \big(\gamma V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t)\big)] \\
&= E_{\tau \sim p(\tau;\theta^{'})}[\sum_{t=1}^{\infty} \gamma^t r_t] + E_{\tau \sim p(\tau;\theta^{'})}[\sum_{t=0}^{\infty} \gamma^t \big(\gamma V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t)\big)] \\
&= E_{\tau \sim p(\tau;\theta^{'})}[\sum_{t=0}^{\infty} \gamma^t \big(r_t + \gamma V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t)\big)] \\
&= E_{\tau \sim p(\tau;\theta^{'})}[\sum_{t=0}^{\infty} \gamma^t A_{\pi_\theta}(s_t, a_t)]
\end{aligned}$$

where $\gamma^0 = 1$ and r_0 is a constant. So, we get

$$J(\theta^{'}) - J(\theta) = E_{\tau \sim p(\tau;\theta^{'})}[\sum_{t=0}^{\infty} \gamma^t A_{\pi_\theta}(s_t, a_t)]$$

where the expectation is under the policy $\pi_{\theta^{'}}$ and the Advantage Function is under π_θ. We let $\rho_\pi(s)$ be the discounted visitation frequency

$$\rho_\pi(s) = \big(P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \cdots\big)$$

and re-express the difference to sum over states, getting

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} J(\theta^{'}) - J(\theta) &= \sum_{t=0}^{\infty} \sum_{s \in \mathcal{S}} P_{\pi_{\theta^{'}}}(s_t = s) \sum_{a \in \mathcal{A}} \pi_{\theta^{'}}(s,a) \gamma^t A_{\pi}(s,a) \\ &= \sum_{s \in \mathcal{S}} \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta^{'}}}(s_t = s) \sum_{a \in \mathcal{A}} \pi_{\theta^{'}}(s,a) A_{\pi}(s,a) \\ &= \sum_{s \in \mathcal{S}} \rho_{\pi_{\theta^{'}}}(s) \sum_{a \in \mathcal{A}} \pi_{\theta^{'}}(s,a) A_{\pi}(s,a) \end{aligned}$$

Any policy such that $\sum_{a \in \mathcal{A}} \pi_{\theta^{'}}(s,a) A_{\pi}(s,a) \leqslant 0$ is guaranteed to reduce J. However, there will be some states s for which the expected advantage is positive. One way around is to locally approximate J as follows:

$$L(\theta^{'}) - J(\theta) = \sum_{s \in \mathcal{S}} \rho_{\pi_{\theta}}(s) \sum_{a \in \mathcal{A}} \pi_{\theta^{'}}(s,a) A_{\pi}(s,a)$$

where $L(\cdot)$ uses the visitation frequency $\rho_{\pi_{\theta}}$. We will show that L matches J to the first order. To do so we work with expectation. Expending $\tau \sim p(\tau;\theta^{'})$ and using importance sampling (see Appendix (26.2.2.4)) to modify the distribution, we get

$$\begin{aligned} E_{\tau \sim p(\tau;\theta^{'})}[\sum_{t=0}^{\infty} \gamma^t A_{\pi_{\theta}}(s_t,a_t)] &= \sum_{t=0}^{\infty} E_{s_t \sim p(s_t;\theta^{'})}[E_{a_t \sim \pi_{\theta^{'}}(a_t,s_t)}[\gamma^t A_{\pi_{\theta}}(s_t,a_t)]] \\ &= \sum_{t=0}^{\infty} E_{s_t \sim p(s_t;\theta^{'})}[E_{a_t \sim \pi_{\theta}(a_t,s_t)}[\frac{\pi_{\theta^{'}}(a_t,s_t)}{\pi_{\theta}(a_t,s_t)} \gamma^t A_{\pi_{\theta}}(s_t,a_t)]] \end{aligned}$$

Assuming that $p(s_t;\theta)$ is close to $p(s_t;\theta^{'})$ when π_{θ} is close to $\pi_{\theta^{'}}$, one can approximate the density $p(s_t;\theta^{'})$ in the outer expectation with $p(s_t;\theta)$, getting

$$\sum_{t=0}^{\infty} E_{s_t \sim p(s_t;\theta^{'})}[E_{a_t \sim \pi_{\theta}(a_t,s_t)}[\frac{\pi_{\theta^{'}}(a_t,s_t)}{\pi_{\theta}(a_t,s_t)} \gamma^t A_{\pi_{\theta}}(s_t,a_t)]] \approx \sum_{t=0}^{\infty} E_{s_t \sim p(s_t;\theta)}[E_{a_t \sim \pi_{\theta}(a_t,s_t)}[\frac{\pi_{\theta^{'}}(a_t,s_t)}{\pi_{\theta}(a_t,s_t)} \gamma^t A_{\pi_{\theta}}(s_t,a_t)]]$$

We let the RHS be denoted by $\overline{A}(\theta^{'})$, and then get

$$J(\theta^{'}) - J(\theta) \approx \overline{A}(\theta^{'}) \Longrightarrow \theta^{'} \leftarrow \arg\max_{\theta^{'}} \overline{A}(\theta^{'}) \tag{17.1.20}$$

In the special case where $\theta^{'} = \theta$, we get

$$\overline{A}(\theta) = E_{\tau \sim p(\tau;\theta)}[\sum_{t=0}^{\infty} \gamma^t A_{\pi_{\theta}}(s_t,a_t)]$$

We now want to find some bounds to the distribution change. Kakade et al. [2002] obtained some bounds in the special case of mixture of policies

$$\pi_{new}(s,a) = (1-\alpha)\pi_{old}(s,a) + \alpha\pi^{'}(s,a)$$

Schulman et al. [2015] extended the result to general stochastic policies by replacing α with a distance measure between π and $\pi^{'}$.

Claim 3 *We claim that if π_{θ} is an arbitrary distribution, then $\pi_{\theta^{'}}$ is close to π_{θ} if $|\pi_{\theta^{'}}(a_t,s_t) - \pi_{\theta}(a_t,s_t)| \leqslant \epsilon$ for all s_t.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Following Lemma (24.1.7), we can infer that $\pi_{\theta'}(a_t, s_t)$ takes a different action than $\pi_\theta(a_t, s_t)$ with probability at most ϵ. We write the mass function $p(s_t; \theta^{'})$ as

$$p(s_t; \theta^{'}) = (1-\epsilon)^t p(s_t; \theta) + \big(1 - (1-\epsilon)^t\big) p(s_t; \xi)$$

where $p(s_t; \xi)$ is another distribution, possibly from a mistake. Using the identity $(1-\epsilon)^t \geqslant 1 - \epsilon t$ for $\epsilon \in [0, 1]$, we get

$$\begin{aligned} |p(s_t; \theta^{'}) - p(s_t; \theta)| = \big(1 - (1-\epsilon)^t\big)|p(s_t; \xi) - p(s_t; \theta)| &\leqslant 2\big(1 - (1-\epsilon)^t\big) \\ &\leqslant 2\epsilon t \end{aligned}$$

Given a smooth function $f(\cdot)$, we can therefore bound its expectation as

$$\begin{aligned} E_{p(s_t;\theta')}[f(s_t)] = \sum_{s_t} p(s_t; \theta^{'}) f(s_t) &\geqslant \sum_{s_t} p(s_t; \theta) f(s_t) - |p(s_t; \theta^{'}) - p(s_t; \theta)| \max_{s_t} f(s_t) \\ &\geqslant E_{p(s_t;\theta)}[f(s_t)] - 2\epsilon t \max_{s_t} f(s_t) \end{aligned}$$

Back to $J(\theta^{'}) - J(\theta)$, we get

$$\begin{aligned} &\sum_{t=0}^{\infty} E_{s_t \sim p(s_t;\theta')}[E_{a_t \sim \pi_\theta(a_t, s_t)}[\frac{\pi_{\theta'}(a_t, s_t)}{\pi_\theta(a_t, s_t)} \gamma^t A_{\pi_\theta}(s_t, a_t)]] \geqslant \\ &\sum_{t=0}^{\infty} E_{s_t \sim p(s_t;\theta)}[E_{a_t \sim \pi_\theta(a_t, s_t)}[\frac{\pi_{\theta'}(a_t, s_t)}{\pi_\theta(a_t, s_t)} \gamma^t A_{\pi_\theta}(s_t, a_t)]] - \sum_{t=0}^{\infty} 2\epsilon t C \end{aligned}$$

where C is $O(\frac{r_{\max}}{1-\gamma})$. Simplifying notation we get the bound $\overline{A}(\theta^{'}) - \sum_{t=0}^{\infty} 2\epsilon t C$.
Schulman et al. [2015] got a better bound by using the Kullback-Leibler divergence $D_{K,L}(P||Q) = \sum_y P(y) \log \frac{P(y)}{Q(y)}$. That is, $D_{K,L}(p_1(x)||p_2(x)) = E_{x \sim p_1(x)}[\log \frac{p_1(x)}{p_2(x)}]$. Identifying P to $\pi_{\theta'}(a_t, s_t)$ and Q to $\pi_\theta(a_t, s_t)$, we get

$$|\pi_{\theta'}(a_t, s_t) - \pi_\theta(a_t, s_t)| \leqslant \sqrt{\frac{1}{2} D_{K,L}(\pi_{\theta'}(a_t, s_t)||\pi_\theta(a_t, s_t))}$$

where $D_{K,L}(\pi_{\theta'}(a_t, s_t)||\pi_\theta(a_t, s_t))$ bounds the state marginal difference.

17.1.7.2 An optimisation problem

Applying a constraint on the KL divergence between the new policy and the old one defines the trust region constraint (TRC). Given the relation in Equation (17.1.20), we need to solve the constrained optimisation problem

$$\begin{aligned} &\theta^{'} \leftarrow \arg\max_{\theta'} \overline{A}(\theta^{'}) \\ &\text{such that } D_{K,L}(\pi_{\theta'}(a_t, s_t)||\pi_\theta(a_t, s_t)) \leqslant \epsilon \end{aligned}$$

Thus, the KL divergence is bounded at every point in the state space. For small enough ϵ we are guaranteed to improve the difference $J(\theta^{'}) - J(\theta)$.
One solution to this problem is to follow the Lagrange approach presented in Section (18.5). The Lagrangian is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$L(\theta^{'},\lambda) = \overline{A}(\theta^{'}) - \lambda\Big(D_{K,L}(\pi_{\theta'}(a_t,s_t)||\pi_\theta(a_t,s_t)) - \epsilon\Big)$$

The steps are as follows:

1. Maximise $L(\theta^{'},\lambda)$ with respect to $\theta^{'}$.
2. $\lambda \leftarrow \lambda + \alpha\big(D_{K,L}(\pi_{\theta'}(a_t,s_t)||\pi_\theta(a_t,s_t)) - \epsilon\big)$.

17.1.7.2.1 Recovering the natural policy gradient Alternatively, we can perform a first order Taylor expansion (linearisation) of $\overline{A}(\theta^{'})$ around $\overline{A}(\theta)$, getting

$$\theta^{'} \leftarrow \arg\max_{\theta'} \nabla_\theta \overline{A}(\theta)^\top \Delta\theta$$

where $\Delta\theta = \theta^{'} - \theta$.
Using Equation (17.1.6), we get the gradient

$$\nabla_{\theta'}\overline{A}(\theta^{'}) = \sum_{t=0}^{\infty} E_{s_t\sim p(s_t;\theta)}[E_{a_t\sim\pi_\theta(a_t,s_t)}[\frac{\pi_{\theta'}(a_t,s_t)}{\pi_\theta(a_t,s_t)}\gamma^t\nabla_{\theta'}\log\pi_{\theta'}(a_t,s_t)A_{\pi_\theta}(s_t,a_t)]]$$

In the special case where $\theta^{'} = \theta$, it simplifies to

$$\nabla_\theta\overline{A}(\theta) = \sum_{t=0}^{\infty} E_{s_t\sim p(s_t;\theta)}[E_{a_t\sim\pi_\theta(a_t,s_t)}[\gamma^t\nabla_\theta\log\pi_\theta(a_t,s_t)A_{\pi_\theta}(s_t,a_t)]] = \nabla_\theta J(\theta)$$

which is the normal policy gradient.
So logically, we can wonder if you can just use the gradient. One solution could be to use the gradient ascent as follows:

$$\theta^{'} \leftarrow \arg\max_{\theta'} \nabla_\theta J^\top(\theta)\Delta\theta$$
$$\text{such that } \|\Delta\theta\|^2 \leqslant \epsilon$$

where the update is

$$\theta^{'} = \theta + \frac{\epsilon}{\|\nabla_\theta J(\theta)\|^2}\nabla_\theta J(\theta)$$

Another solution would be to rescale the gradient and consider the problem

$$\theta^{'} \leftarrow \arg\max_{\theta'} \nabla_\theta J^\top(\theta)\Delta\theta$$
$$\text{such that } D_{K,L}(\pi_{\theta'}(a_t,s_t)||\pi_\theta(a_t,s_t)) \leqslant \epsilon$$

Even though the constraints are not the same, we can perform a second order Taylor expansion of the Kullback-Leibler divergence, getting

$$D_{KL}(\pi_{\theta'}||\pi_\theta) \approx \frac{1}{2}(\Delta\theta)^\top F\Delta\theta$$

where F is the Fisher-information matrix

$$E_{\pi_\theta}[\nabla_\theta\log\pi_\theta(s,a)\big(\nabla_\theta\log\pi_\theta(s,a)\big)^\top]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

which can be statistically estimated. Then we get the update

$$\theta^{'} = \theta + \alpha F^{-1} \nabla_\theta J(\theta)$$

We can solve for α while solving $F^{-1}\nabla_\theta J(\theta)$ (see Schulman et al. [2015]). We get

$$\alpha = \frac{2\epsilon}{\big(\nabla_\theta J(\theta)\big)^\top F \nabla_\theta J(\theta)}$$

17.2 Minimising risk in MDP

We have seen so far that the goal of reinforcement learning was to learn how to take actions in order to maximise reward. However, some practical problems require maximising rewards while minimising some measure of risk. As a result, some authors modified the classical Markov decision process (MDP) to account for this requirement. For instance, the variance-penalised Markov decision process (VPMDP) is an MDP following a mean-variance criterion (see Appendix (28.1.1.1)) instead of the classical reward. Risk in MDPs has been studied in the context of exponential utility function. This is called risk-sensitive MDPs. When one is considering the mean-variance criterion, it is called risk-penalised MDPs.
We are briefly going to present a few measure of risk and then discuss one implementation of the variance-penalised MDP.

17.2.1 Measures of risk

In general, we define the problem of minimising risk as

$$\begin{aligned} w_0 &= \arg\min_{w \in \mathbb{R}^p} \Psi(w^\top X) \\ &\quad \text{such that } w^\top I_p = I \text{ and } w^\top \mu = R \end{aligned}$$

where $w \in \mathbb{R}^p$ is the weight vector, $X \in \mathbb{R}^p$ is the input vector (returns), $\mu = E[X]$ is the mean return vector and $\Psi : \mathbb{R} \to \mathbb{R}$ is some measure of risk. The first constraint allows normalisation to 1, and the second constraint is a target return constraint which may or may not be active.
Some examples of measures of risk are:

- $\Psi(w^\top X) = w^\top \Sigma w$ is the classical mean-variance model of Markowitz (see Appendix (28.1.1.1)). μ and Σ are respectively the mean and the covariance matrix of X.
- $\Psi(w^\top X) = CVaR(-w^\top X; \beta)$ where $\beta \in (\frac{1}{2}, 1)$ and

  $$CVaR(-w^\top X; \beta) = \min_\alpha \alpha + \frac{1}{1-\beta} E[\big(-w^\top X - \alpha\big)^+]$$

  is the conditional value-at-risk (CVaR) formulation for the level $100(1-\beta)\%$ (see Rockafellar et al. [2000]) (see details in Appendix (28.1.2.3)).

As discussed in Appendix (28.1.2), variance is a flawed measure of risk when the random variables X are non-normal. The agent (or decision maker) should prefer high average returns, lower variance or standard deviation, positive skewness, and lower kurtosis. Thus, alternative measures of risk should be considered.
In practice, we do not know the true distribution of X, but we have access to past (or simulated) data $X = [X_1, ..., X_n]$. Assuming that the data is i.i.d., the standard approach is to solve the empirical risk minimisation (ERM) problem (see Section (10.1.1))

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}\widehat{w}_n &= \arg\min_{w\in\mathbb{R}^p} \widehat{\Psi}_n(w^\top X) \\ &\quad \text{such that } w^\top I_p = I \text{ and } w^\top \widehat{\mu}_n = R\end{aligned}$$

where $\widehat{\Psi}_n(\cdot)$ is the sample average estimate of the risk function and $\widehat{\mu}_n = \frac{1}{n}\sum_{i=1}^n X_i$ is the sample average return vector.
Given the above measures of risk, the ERM becomes

- The mean-variance model:

$$\begin{aligned}\widehat{w}_{n,MV} &= \arg\min_{w\in\mathbb{R}^p} w^\top \widehat{\Sigma}_n w \\ &\quad \text{such that } w^\top I_p = I \text{ and } w^\top \widehat{\mu}_n = R\end{aligned}$$

  where $\widehat{\Sigma}_n$ is the sample covariance of the matrix X.

- The CVaR model:

$$\begin{aligned}\widehat{w}_{n,CV} &= \arg\min_{w\in\mathbb{R}^p} \widehat{C}VaR(-w^\top X;\beta) \\ &\quad \text{such that } w^\top I_p = I \text{ and } w^\top \widehat{\mu}_n = R\end{aligned}$$

  where

$$\widehat{C}VaR(-w^\top X;\beta) = \min_{\alpha\in\mathbb{R}} \alpha + \frac{1}{n(1-\beta)}\sum_{i=1}^n \left(-w^\top X_i - \alpha\right)^+$$

  is the sample average estimator for $CVaR(-w^\top X;\beta)$.

Asymptotically, as the number of observation n goes to infinity, the sample average approximation (SAA) solution $\widehat{w}_n$ converges in probability to w. However, this SAA method can be highly unstable. Ban et al. [2016] proposed a performance-based regularisation (PBR) approach to improve upon the performance of the SSA by constraining the sample variance. The idea being to reduce the chance of solution being chosen by misleadingly high in-sample performance. The model is

$$\begin{aligned}\widehat{w}_{n,PBR} &= \arg\min_{w\in\mathbb{R}^p} \widehat{\Psi}_n(w^\top X) \\ &\quad \text{such that } w^\top I_p = I \text{ and } w^\top \widehat{\mu}_n = R \\ &\quad SVar(\widehat{\Psi}_n(w^\top X)) \leqslant U_1 \text{ and } SVar(w^\top \widehat{\mu}_n) \leqslant U_2\end{aligned}$$

where $SVar(\cdot)$ is the sample variance operator, and U_1, U_2 are parameters controlling the degree of regularisation.

17.2.2 The variance-penalised MDP

The variance-penalised MDP problem with infinite-time horizon has been studied for asymptotic and one-step variance. The objective is the expected long-run reward minus a constant times the variance (see Equation (28.1.1)), which is used as a measure of risk. That is, the policy simultaneously maximises the expected rewards and minimises the variability in the reward. Several authors used mathematical programming to study this problem (see Sobel [1982]). In the case of asymptotic variance, Gosavi et al. [2010] proposed a Bellman equation and a value iteration (VI) algorithm. Gosavi [2006] considered a one-step variance as the risk measure.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In the case of finite-time horizon, Collins [1997] considered the asymptotic variance together with a terminal reward. Gosavi [2010] extended this approach by letting the rewards be non-zero in every state. The author used one-step variance as a measure of risk within the dynamic programming framework of finite horizon. The stochastic shortest path (SSP) algorithm of Bertsekas et al. [1991] shares an intimate relationship with the finite horizon MDP when the transition probabilities are known. The author considered a special case of this SSP by viewing the stage as an extra component in the state and assuming that the number of stages is fixed. Otherwise, when the transition probabilities are unknown, a Q-learning algorithm is proposed. Gosavi [2014] presented a description of the algorithms used both in the infinite-time and finite-time horizon MDP. We briefly review the algorithm for the finite-time horizon MDPs.

17.2.2.1 Bellman equations

In the SSP problem the transition probabilities are stationary and modelled with a Markov chain. We want to maximise the expected total reward accumulated till the finite horizon time. $\mathcal{A}(i)$ is the finite set of actions permitted in state i, $d(i)$ is the action chosen in state i when policy $\hat{d}$ is followed, where $\bigcup_{i\in\mathcal{S}}\mathcal{A}(i)=\mathcal{A}$. Further, $r:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\to\mathbb{R}$ denotes the one-step reward and $p:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\to[0,1]$ is the associated transition probability.
For the discounted reward MDP, the Q-value function $Q:\mathcal{S}\times\mathcal{A}\to\mathbb{R}$ is given by

$$Q(i,a)=\sum_{j\in\mathcal{S}}p(i,a,j)\big[r(i,a,j)+\gamma\max_{b\in\mathcal{A}(j)}Q(j,b)\big]\,,\forall(i,a)$$

In the case of a finite-horizon MDP problem, there are a finite number of stages, N, through which the system can pass before it enters an absorbing state (reward-free state). Thus, the state-action pair (i,a) is replaced by the state-stage-action triple (i,n,a) where n is the stage index taking values in the set $\mathcal{N}=\{1,2,...,N\}$. Further, we assume that there is no action performed in stage $(N+1)$ and that, irrespective of the state or action, the Q-value in that stage will be zero. The starting state, when $n=1$, is assumed known with certainty. For all $i\in\mathcal{S}$, $n\in\mathcal{N}$, and all $a\in\mathcal{A}(i,n)$, the Q-value function is

$$Q(i,n,a)=\sum_{j\in\mathcal{S}}p(i,n,a,j,n+1)\big[r(i,n,a,j,n+1)+\max_{b\in\mathcal{A}(j,n+1)}Q(j,n+1,b)\big]\,,\forall(i,a)$$

where $Q(j,N+1,b)=0$ for all $i\in\mathcal{S}$ and $b\in\mathcal{A}(j,N+1)$. The policy π defined by $\pi(i,n)\in\arg\max_{a\in\mathcal{A}(i,n)}Q(i,n,a)$ is an optimal policy for the MDP.

17.2.2.2 The known model

The immediate expected reward earned in state i when action a is chosen is defined as

$$\overline{r}(i,a)=\sum_{j\in\mathcal{S}}p(i,a,j)r(i,a,j)$$

We let $v:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\to\mathbb{R}$ denotes the one-step immediate variance, defined for $(i,j)\in\mathcal{S}$ and $a\in\mathcal{A}(i)$, as

$$v(i,a,j)=\big(r(i,a,j)-\overline{r}(i,a)\big)^2$$

For $i\in\mathcal{S}$ and $a\in\mathcal{A}(i)$, we have the mean variance

$$\overline{v}(i,a)=\sum_{j\in\mathcal{S}}p(i,a,j)v(i,a,j)$$

We make the following assumptions:

***Assumption* 17.2.1** *Every policy is proper: under every policy the system inevitably reaches a terminal state which is unique.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

***Assumption* 17.2.2** *The starting state of the system is known and fixed.*

***Assumption* 17.2.3** *The terminal state of the system is unique and absorbing, and it generates no reward.*

The long-run expected value of the total reward earned by a policy $\widehat{\pi}$ starting at state i over N stages is

$$R_{\widehat{\pi}}(i) = E_{\widehat{\pi}}[\sum_{n=1}^{N} \overline{r}(s_n, \pi(s_n))|s_1 = i]$$

where s_n is the state of the system in the nth stage.
The long-run one-step variance in the immediate rewards of a policy $\widehat{\pi}$ starting at state i over N stages is

$$V_{\widehat{\pi}}(i) = E_{\widehat{\pi}}[\sum_{n=1}^{N} \overline{v}(s_n, \pi(s_n))|s_1 = i]$$

The objective function is

$$\max R(i) - \lambda V(i)$$

for a given starting state i.
We can alter the reward structure of the SSP by adding a variance penalty. We define

$$w(i, n, a, j, n+1) = r(i, n, a, j, n+1) - \lambda\big(r(i, n, a, j, n+1) - \overline{r}(i, n, a)\big)^2$$

and replace $r(i, n, a, j, n+1)$ in the original SSP by $w(i, n, a, j, n+1)$ defined above. This transformation allows the author to apply the theory of SSP to solve the one-step variance penalised MDP.
We let $J(i, n)$ denotes the element of the value function for state i in stage n of the finite horizon MDP where (i, n) defines the state. J is a matrix but we can map its element into a vector $\widetilde{J}$ made of $|\mathcal{K}| = |\mathcal{S}| \cdot N$ elements.
The steps of the algorithm are as follows:

1. Select $\epsilon > 0$. Set $k = 0$. Set arbitrary values (such as 0) to $J_k(i, t)$ for all $i \in \mathcal{S}$ and $t = 1, 2, ..., N+1$. Set $n = 1$. Let i_* be the starting state.

2. The update is performed for all $i \in \mathcal{S}$, unless $n = 1$ in which case it is only performed for i_*.

$$J_{k+1}(i, n) = \max_{a \in \mathcal{A}(i,n)} \Big[\sum_{j \in \mathcal{S}} \widetilde{p}\big[w(i, n, a, j, n+1) + J_k(j, n+1)\big]\Big]$$

   where

$$\widetilde{p} \equiv p(i, n, a, j, n+1)$$

3. Increment n by 1. If $n = N+1$ go to step [4]. Else, return to step [2].

4. Check if $\|\widetilde{J}_{k+1} - \widetilde{J}_k\|_\infty < \epsilon$. If true, go to step [5]. Else, set $n = 1$, increment k by 1 and return to step [2].

5. Determine the optimal action in each state-stage pair (i, n) as follows:

$$\arg \max_{a \in \mathcal{A}(i,n)} \Big[\sum_{j \in \mathcal{S}} \widetilde{p}\big[w(i, n, a, j, n+1) + J_k(j, n+1)\big]\Big]$$

   and stop.

Note, to make sure that the terminal state in stage $(N+1)$ is reward-free, $J(i, N+1)$ is never updated for all i and remains at zero.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.2.2.3 The unknown model

When the transition probabilities are not known, the author used the Q-learning algorithm (see details in Section (17.1.3)). It is assumed that a simulator of the system is available. The algorithm uses uses two different step-sizes, α and β that satisfy the relationship

$$\lim_{k\to\infty}\frac{\beta_k}{\alpha_k}=0$$

An example of step-sizes satisfying this rule is $\alpha_k=\frac{A}{B+k}$ and $\beta_k=\frac{C}{k\log k}$ where A,B,C are constants. The above condition implies that updating with β represents a slower time-scale, while α corresponds to a faster time scale.
The steps of the algorithm are as follows:

1. For all (l,u), where $l\in\mathcal{S}$, $n=1,2,...,N+1$ and $u\in\mathcal{A}(l,n)$, set

$$Q(l,n,u)=0 \text{ and } \widetilde{r}(l,n,u)=0$$

   where $\widetilde{r}(l,n,u)$ is the estimate of the immediate expected reward in state (l,n) when action u is chosen. Set the number of state changes k to zero. Set the maximum number of iterations for which the algorithm is run $k_{\max}$ to a sufficiently large number. The algorithm run iteratively between step [2] and [6]. The starting state is i_* and $n=1$.

2. Let i be the current state and n be the current stage. Select action a with probability $\frac{1}{|\mathcal{A}(i,n)|}$ or some other rule (Boltzmann selection rule).

3. Simulate action a. Let the next state be j in stage $(n+1)$. Increment k by 1.

4. Update $Q(i,n,a)$ as follows:

$$Q_{k+1}(i,n,a)\leftarrow(1-\alpha_k)Q_k(i,n,a)+\alpha_k\Big[w(i,n,a,j,n+1)+\max_{b\in\mathcal{A}(j,n+1)}Q_k(j,n+1,b)\Big]$$

   where

$$w(i,n,a,j,n+1)=r(i,n,a,j,n+1)-\lambda\big[r(i,n,a,j,n+1)-\widetilde{r}(i,n,a)\big]^2$$

5. Update $\widetilde{r}(i,n,a)$ as follows

$$\widetilde{r}(i,n,a)\leftarrow(1-\beta_k)\widetilde{r}(i,n,a)+\beta_k r(i,n,a,j,n+1)$$

6. If $k=k_{\max}$ go to step [7] otherwise set $n=n+1$. If $n=N+1$, set $i=i_*$, $n=1$ and return to step [2]. Else, set $i\leftarrow j$ and return to step [2].

7. For each $l\in\mathcal{S}$ and $n=1,2,...,N$, select $d(l,n)\in\arg\max_{b\in\mathcal{A}(l,n)}Q(l,n,b)$. The policy generated by the algorithm is $\widehat{d}$. Stop.

Note, we make sure that the terminal state in stage $(N+1)$ is reward-free because $Q(i,N+1,a)$ is never updated for all (i,a) pairs and remains at zero. The author proved that this algorithm converges to the optimal solution of the one-step variance penalised finite horizon MDP.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

17.3 Continuous state spaces

The problem of approximating optimal policies for finite state spaces described in Section () can not easily be generalised to the continuous case. In the latter, RL attempt to approximate the optimal value function V^* directly from a given sample of trajectories from the MDP. For instance, the kernel-based RL (see Ormoneit et al. [2002]) guarantees convergence to the optimal value function. Jong et al. [2006] provided a modification where a continuous MDP is understood to be equivalent to finding the exact value of an approximate finite MDP, while still maintaining the same convergence. We briefly describe this approach.
We assume we have a sample with n transitions $\left(s^{(i)}, \widehat{s}^{(i)}, a^{(i)}\right)$ where we go from state $s^{(i)}$ to $\widehat{s}^{(i)}$ as a result of action $a^{(i)}$ obtained from the MDP $M = (\mathcal{S}, \mathcal{A}, \mathcal{T}, R)$ with continuous state space $\mathcal{S}$, discrete action space $\mathcal{A}$, transition probability $\mathcal{T}$ and reward function R. The authors used such a sample to approximate M with a finite MDP $\tilde{M} = (\widehat{\mathcal{S}}, \mathcal{A}, \tilde{\mathcal{T}}, \tilde{R})$ where $\widehat{\mathcal{S}} = \{\widehat{s}^{(i)}\}$ is the set of successor states. Given a kernel function ϕ (see Appendix (23.2.1.4)), a bandwidth b and premetric d, then $\tilde{\mathcal{T}}$ and $\tilde{R}$ are given by

$$\tilde{\mathcal{T}}(s, a, \widehat{s}^{(i)}) = \begin{cases} \frac{1}{Z_{s,a}} \phi(\frac{d(s,s^{(i)})}{b}) \text{ if } a^{(i)} = a \\ 0 \text{ otherwise} \end{cases}$$

and

$$\tilde{R}(s, a) = \frac{1}{Z_{s,a}} \sum_{i|a^{(i)}=a} \phi(\frac{d(s, s^{(i)})}{b}) R(s^{(i)}, a^{(i)})$$

where

$$Z_{s,a} = \sum_{i|a^{(i)}=a} \phi(\frac{d(s, s^{(i)})}{b})$$

Some standard choices is to use the Euclidian distance metric d and a Gaussian kernel ϕ. In that setting, the exact solution of $\tilde{M}$, $\tilde{V}^*$ converges to the exact solution of M, V^* as the number of samples increases.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Part V

Optimisation

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 18

Introduction to constrained optimisation

Every field of science and engineering makes use of search algorithms, also known as optimisation algorithms, which seek to automatically find a high-quality solution or set of high-quality solutions amongst a large space of possible solutions. In mathematics and computer science, an optimisation problem is the problem of finding the best solution from all feasible solutions. Optimisation problems can be divided into two categories depending on whether the variables are (1) continuous or (2) discrete. In an optimisation problem with discrete variables (discrete optimisation), we are looking for an object such as an integer, permutation or graph from a finite (or possibly countably infinite) set. Problems with continuous variables include constrained problems and multimodal problems. In this chapter we are going to focus on the latter. In such problems, we try to optimise (maximise or minimise) some quantity, while satisfying some constraints. Optimisation problems are of particular interest in the area of Operations Research and they appear in many real situations. It has important applications in several fields, including artificial intelligence, machine learning, auction theory, and software engineering.

See textbooks by Bertsekas [1996] [2015], Bazaraa et al. [1993], Nocedal et al. [1999], Boyd et al. [2004], Sun et al. [1999], among other.

18.1 Introduction

18.1.1 Defining the problem

18.1.1.1 The constrained problem

We consider a system with the real-valued properties

$$g_m \text{ for } m = 0, .., P-1$$

making the objectives of the system to be optimised. Given a N-dimensional vector of real-valued parameter $X \subset \mathbb{R}^N$ the optimisation problem can always be written as

$$\min f_m(X)$$

where $f_m(\cdot)$ is a function by which g_m is calculated and where each element $X(i)$ of the vector is bounded by lower and upper limits $L_i \leqslant X(i) \leqslant U_i$ which define the search space $\mathcal{S}$. Depending on the linearity or non-linearity of the objective function we can distinguish between linear and non-linear optimisation. The set $\mathcal{S}$ is called the feasible set. We speak of discrete optimisation problems if $\mathcal{S}$ is finite, or if it consists of grid points with integer coordinates.
We follow Lueder [1990] who showed that all functions $f_m(\cdot)$ can be combined in a single objective function $H : X \subset \mathbb{R}^N \to \mathbb{R}$ expressed as the weighted sum

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$H(X) = \sum_{m=1}^{P} w_m f_m(X)$$

where the weighting factors w_m define the importance of each objective of the system. Hence, the optimisation problem becomes

$$\min H(X)$$

so that all the local and global minima (when the region of eligibility in X is convex) can be found. However, most problems involves a single objective function, so that the optimisation function simplifies. Most complex search problems such as optimisation problems are constrained numerical problem (CNOP) more commonly called general nonlinear programming problems with constraints given by

$$g_i(X) \leqslant 0 \,,\, i = 1, .., p$$
$$h_j(X) = 0 \,,\, j = 1, .., q$$

Equality constraints are usually transformed into inequality constraints by

$$|h_j(X)| - \epsilon \leqslant 0$$

where ϵ is the tolerance allowed.
Given the search space $\mathcal{S} \subset \mathbb{R}^N$, we let $\mathcal{F}$ be the set of all solutions satisfying the constraints of the problems, called the feasible region. It specifies the solutions X which are permitted (or feasible). It is defined by the intersection of $\mathcal{S}$ and the set of $p + q$ additional constraints. At any point $X \in \mathcal{F}$, the constraints $g_i(\cdot)$ that satisfy $g_i(X) = 0$ are active constraints at X while equality constraints $h_j(\cdot)$ are active at all points of $\mathcal{F}$.

18.1.1.2 Some examples

We let $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$, $h : \mathbb{R}^n \to \mathbb{R}^l$ and consider the optimisation problem

$$\min_x f(x) \text{ such that } g(x) \leqslant 0 \,,\, h(x) = 0 \tag{18.1.1}$$

As discussed above, any equality constraint $h_i(x)$ can be transformed into two inequality constraints

$$h_i(x) \geqslant 0 \text{ and } -h_i(x) \geqslant 0$$

such that

$$\min_x f(x) \text{ such that } g(x) \leqslant 0 \tag{18.1.2}$$

is sufficiently general.
Some examples of optimisation problems are

- Linear regression:

$$\min_w \|xw - y\|^2$$

- Classification (logistic regression or SVM)

$$\min_w \sum_{i=1}^{n} \log\left(1 + e^{-y_i x_i^\top w}\right)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

or

$$\|w\|^2 + C\sum_{i=1}^{n}\xi_i \text{ such that } \xi_i \geqslant 1 - y_i x_i^\top w\,,\, \xi_i \geqslant 0$$

- Maximum likelihood estimation:

$$\max_{\theta}\sum_{i=1}^{n}\log p_\theta(x_i)$$

- k-,means

$$\min_{\mu_1,\ldots,\mu_k} J(\mu) = \sum_{j=1}^{k}\sum_{i\in C_j}\|x_i - \mu_j\|^2$$

Some important examples in machine learning are

- SVM loss:

$$f(w) = \big(1 - y_i x_i^\top w\big)^+$$

- Binary logistic loss:

$$f(w) = \log\big(1 + e^{-y_i x_i^\top w}\big)$$

18.1.2 Search techniques

Nearly all science and engineering fields use search algorithms, which automatically explore a search space to find high-performing solutions. Following Section (18.1.1), we let the vector $X \in \mathbb{R}^N$ be a particular point within a search space of all such vectors. We are concerned about finding an algorithm that searches this space with the minimum amount of effort. A large number of well established search techniques exists such as calculus-based methods, exhaustive search, and random (see Goldberg [1989]). These methods are represented in Figure (18.1).

Figure 18.1: Search techniques

They are summarised as follows:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1. Enumerative techniques: They require a finite search space, or a discretised infinite search space of possible values for the ojective function. Then, they search every possible point one point at a time to find the optimum. While they can be simple to implement, the number of possible points may be too large for direct search. In some cases, such as game playing (see Uiterwijk et al. [1989]), it is possible to curtail the search so that points in regions which cannot contain the solution are not checked.

2. Calculus based techniques: They treat the search space as a continuous multi-dimensional function and look for maxima (or minima) using its derivative. These methods come in two categories:

   - Indirect methods: They take the gradient of the objective function, set it equal to zero, and solve the resulting set of equations. They use the fact that at the extrema the function's derivative is zero. Where the function is smooth the volume of space where its derivative is zero can be a very small sample of the whole search space.
   - Direct calculus techniques: They follow the direction of the gradient towards a local optimum value. For example, the Newton method uses the gradient/function values to estimate the location of nearby extrema. These techniques, and others, are known as Hill Climbing techniques (or Gradient Ascent) because they estimate where a maximum (i.e. hill top) lies, move to that point, make a new estimate, move to it and so on until they reach the top of the hill. These techniques can be used upon well behaved problems or problems which can be transformed to become well behaved.

3. Stochastic search techniques: They use information from the search so far to guide the probabilistic choice of the next point(s) to try. They are general in their scope, being able to solve some very complex problems that are beyond the abilities of either enumerative or calculus techniques. Some of these techniques are:

   - Simulated annealing searches for minimum energy states using an analogy based upon the physical annealing process, where large soft low energy crystals can be formed in some metals (for example copper) by heating and then slow cooling (see Kirkpatrick et al. [1983]).
   - Evolutionary algorithms are based upon Darwin's natural selection theory of evolution, where a population is progressively improved by selectively discarding the worse and breeding new children from the better (see Section (19.1)).
     - In Evolutionary Strategies points in the search space are represented by a vector of real values. Each new point is created by adding random noise to the current one. If the new point is better search proceeds from it, if not the older point is retained.
     - In contrast, a genetic algorithm represents points in the search space by a vector of discrete (typically) bit values. Each new child is produced by combining parts of the bit vector from each parent. This is analogous to the way chromosomes of DNA (which contains the inherited genetic material) are passed to children in natural systems. When programming neural networks, calculus based search techniques such as back propagation are often used. This is possible because the search space has been transformed (and simplified) to be smooth. When the search has succeeded the neural network is said to have been trained, that is, a suitable combination of connection weights has been found. In that setting, a program (a neural network) has been automatically generated by searching for it.

Enumerative techniques are the least efficient optimisation methods for practical problems since the search spaces are too vast to test every possibility one at a time. The calculus based techniques only search for local optima, making them useless when we do not know the neighbourhood of the global optimum or if there are other local optima nearby. Further, they require the existence of derivatives, which is hardly the case in practical applications. Even though the random techniques are faster than an exhaustive search, they can not do better.
Many practical problems have objective functions that are non-differentiable, non-continuous, non-linear, noisy, multi-dimensional and have many local minima. One advantage of genetic algorithms over the traditional methods is that they progress from a population of candidate solutions instead of a single value, which greatly reduces the likelihood of

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

finding a local optimum instead of a global one. Further, they do not require extra information, such as the derivatives, which is unrelated to the values of the possible solutions. Their search mechanism relies on the computation of fitness function, which makes them well suited to search noisy and nonlinear spaces. However, it can be time consuming.
In Chapter (18) we focus on describing the calculus based techniques and introduce some hints at dynamic programming. We will describe the stochastic techniques in Chapter (19).

18.1.3 Discrete optimisation

We have defined the optimisation problems in Section (18.1.1). We are now going to briefly introduce discrete optimisation problems, which consist in finding an optimal object from a set of objects. In an optimisation problem with discrete variables (discrete optimisation), we are looking for an object such as an integer, permutation or graph from a finite (or possibly countably infinite) set. There is an objective function to be minimised, with a very large discrete configuration space. The number of elements in that space is factorially large, so that they can not be explored exhaustively. Further, in a discrete set, we can not rely on the notion of continuing downhill in a favourable direction.
Among the different branches of discrete optimisation, combinatorial optimisation refers to problems on graphs, matroids and other discrete structures. Note, some combinatorial optimisation problems can be solved with integer programs and vice versa. The domain of those optimisation problems has discrete set of feasible solutions such that exhaustive search is not tractable and one is left to finding the best solution.
Some examples of combinatorial optimisation problems are shortest paths and shortest path trees, flows and circulations, spanning trees, matching and matroid problems. In graph theory, the shortest path problem consists in finding a path between two vertices in a graph such that the sum of the weights of its constituent edges is minimised. In this section, we will focus on the travelling salesman problem (TSP), which consists in finding the shortest path going through every vertex exactly once, and returning to the start. Another example is the job assignment problem, where we have more machine available than job to be done. We must allocate the jobs to satisfy demand while minimising the total cost, that is, find the minimum cost assignment of tasks to machines. Another example is the $(0,1)$-knapsack problem where we are given a set of N items and item j has value c_j and size a_j. We are also given a capacity b. We must put selected items in a knapsack of capacity b so as to maximise their total value.

18.1.3.1 Defining the TSP

Some common problems involving combinatorial optimisation are the travelling salesman problem (TSP) and the minimum spanning tree problem (MST). The travelling purchaser problem and the vehicle routing problem are both generalisations of TSP.
The TSP asks the following question: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city? Thus, given a graph, one needs to search the space of permutations to find an optimal sequence of nodes with minimal total edge weights (tour length).

We can formalise TSP in the language of graph theory (see Section (13.1.1.2)). In a graph model of TSP, nodes are places, and every edge $e \in E$ is a street connecting two places and has a certain length c_e. The goal is to find a shortest Hamiltonian cycle in G. In the Asymmetric TSP (ATSP), travelling from i to j and from j to i can incur different costs (for example the presence of one-way streets.) To distinguish the usual TSP from ATSP, we call it sometimes the Symmetric TSP (STSP). A graph-theoretic formulation of ATSP has to use directed graphs. It is the problem of finding the shortest directed Hamiltonian cycle in a directed graph, that is, a Hamiltonian cycle that also respects the directions of arcs. The term Euclidean TSP refers to the special case of STSP where the nodes are points in the plane, and distances are the Euclidean distances. We speak of Metric TSP if the edge lengths satisfy the triangle inequality

$$\forall i,j,k : c_{ik} \leqslant c_{ij} + c_{jk}$$

which is a rather natural condition. Note that the feasible set $\mathcal{S}$ of TSP is already clear from specification: $\mathcal{S}$ is the set of all Hamiltonian cycles of G. We can say that $\mathcal{S}$ is implicitly given by this description, whereas an explicit list of

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

all solutions in $\mathcal{S}$ would be prohibitively large. A graph with N nodes has up to $\frac{(N-1)!}{2}$ different Hamiltonian cycles. Due to this combinatorial explosion, a naive exhaustive search for an optimal solution takes too much time. Thus, we need to find more intelligent methods.

As an example, we consider 25 cities with $\frac{25!}{2}$ possible solutions. It makes it very difficult for an exhaustive search algorithm to find the solution as this number is far too large for testing every single candidate even with a smaller number of cities. The cities with one solution are represented in Figure (18.2).

Figure 18.2: TSP cities and solution

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

A particular solution is described by a vector. A better approach to exhaustive search can be obtained by performing genetic operations such as crossover and mutation. See Figure (18.3). We describe this approach in Section (19.1.3).

Figure 18.3: TSP new solution

18.1.3.2 Some solution methods

Some solution methods are categorised as

- Exact methods: They are guaranteed to find the optimal solution in finite computational time.
- Heuristic methods: They have no guarantee for optimality but they are based on reasonable, justifiable principles or rules of thumb, leading to good or near-optimal solutions. These methods are often easy to implement and efficient.

18.1.3.2.1 Exact methods Brute force method enumerate all the solutions, check feasibility, calculate objective values and keep the best. This is an exact method but suitable only for very small instances.
The most common general strategy for discrete optimisation problems is the Branch and bound (BB) method. This exact method is suitable for a general integer programming problems and can be tailored for different problem classes. It is based on a divide and conquer principle: the problem is partitioned or divided into sub-problems and bounds for the objective value are calculated for each sub-problem. The branching produces a tree structure. The complete enumeration is avoided by pruning or discarding the non-optimal branches by means of comparing the bounds.

18.1.3.2.2 Heuristic methods In general, heuristic methods are preferred over exact methods in the following situation:

- The problem is so large or high dimensional that the exact methods might take too long.
- The exact methods, if they exist, require the problem to be coded in a format that is too complicated, extensive or time consuming.
- No codes are available for exact algorithms, programming would be too tedious, time consuming, and expensive.
- A feasible and functional solution is needed quickly.
- Problem data is uncertain, inaccurate or fuzzy so that exact solution is not of much worth.

The evaluation criteria for heuristics are:

- Quality of solution: The closeness of the value of the objective function to the optimum. The dependence of the quality on the size or structure of the problem, accuracy of problem parameters.
- Cost of solution: The time it takes to get the solution. The work, time and other resources required: Modelling, programming, computing time, space requirements.

We define an approximation algorithm as an algorithm that produces a solution such that its objective function value is within specified distance from the optimum.
We define an instance of a problem as an individual case of a particular problem, where the input parameters are fixed.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 18.1.1 *g-approximation*
Let $x^*(I)$ denote an optimal solution of an instance I of a particular problem. An approximation algorithm A produces a g-approximation, if the following condition holds for the solution $x(I)$:

$$\frac{|f(x(I)) - f(x^*(I))|}{|f(x^*(I))|} \leqslant g(I)$$

for all instances I.

For minimisation with a positive objective function, the condition can be written

$$f(x(I)) \leqslant \big(1 + g(I)\big) f(x^*(I))$$

The function g is usually dependent on the size of the instance I. An ideal approximation algorithm would have a constant bound $g(I) = \epsilon$ that can be made arbitrarily small.
Some general principle for the design of heuristic methods are:

- Construction: The solution is built up piece by piece, by adding elements to a partial arrangement. For example a TSP tour is constructed from a chosen initial vertex by adding vertices or arcs one by one according to a specified principle.
- Improvement: An initial feasible solution (for example a TSP tour) is modified in order to achieve an improvement in the objective value (a shorter tour). The trial solutions are usually generated from specified neighbourhood of the current solution. This is called a local search or neighbourhood search.
- Partitioning: The problem is partitioned into smaller parts which are solved separately and the final solution is assembled from the parts. For example, in TSP the vertex set of a graph is partitioned by a clustering method. The TSP tours of the sub-graphs are constructed and finally joined to form a total solution.
- Decomposition: The problem is solved in sub-sequential phases or stages. The solution of the current phase is based on the previous one. This strategy is used in dynamic programming (DP).
- Random search: Trial solutions can be generated randomly or the current solution can be modified randomly, using appropriate probability distributions.
- Mixed strategies: Combinations of different principles:
  - Construct an initial solution and try to improve it.
  - If local improvement is impossible, make a random modification.
  - Generate a random initial solution and improve it. Repeat. The probability of finding the optimum solution increases with the number of repetitions.

It is sometimes possible to estimate the cost of reaching the goal state from an intermediate state. If the estimate is guaranteed to be an underestimate, the heuristic is called admissible. They have desirable properties in terms of optimality of solution.

18.2 Conditions on optima

In order to describe the calculus based techniques we need to identify the general transformations that convert the vast discrete space into a continuous one, which is sufficiently well behaved to be searched by calculus based techniques. To do so, we need to define the conditions for obtaining local and global optima.
We let $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$, $h : \mathbb{R}^n \to \mathbb{R}^l$ and consider the optimisation problem in Equation (18.1.1).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

18.2.1 Conditions for a local optimum

We assume no constraints.

18.2.1.1 Local minimum

Necessary and sufficient conditions for a local optimum: x^* is a local minimum of $f(x)$ if and only if

1. f has a zero gradient at x^*:
$$\nabla_x f(x^*) = 0$$
2. and the Hessian of f at x^* is positive semi-definite:
$$v^\top \big(\nabla^2 f(x^*)\big) v \geqslant 0 \,, \forall v \in \mathbb{R}^n$$
where
$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{pmatrix}$$

18.2.1.2 Local maximum

Similarly, x^* is a local maximum of $f(x)$ if and only if

1. f has a zero gradient at x^*:
$$\nabla_x f(x^*) = 0$$
2. and the Hessian of f at x^* is negative semi-definite:
$$v^\top \big(\nabla^2 f(x^*)\big) v \leqslant 0 \,, \forall v \in \mathbb{R}^n$$

18.2.2 Conditions for constrained local optimum

We explore the condition to decrease the cost function in the case of both an equality constraint and an inequality constraint. We use the notion of steepest descent described in Appendix (18.3.1.1).

18.2.2.1 The equality constraint

At any point x the direction of the steepest descent of the cost function $f(x)$ is given by $\Delta x = -\nabla_x f(x)$. Thus, to move δx from x such that $f(x + \delta x) < f(x)$ we must have

$$\delta x \cdot \big(-\nabla_x f(x)\big) > 0$$

We then consider the conditions to remain on the constraint surface: Normals to the constraint surface are given by $\nabla_x h(x)$. Note, the direction of the normal is arbitrary as the constraint can be imposed as either $h(x) = 0$ or $-h(x) = 0$. To move a small δx from x and remain on the constraint surface we need to move in a direction which is orthogonal to $\nabla_x h(x)$.
More generally, for x_F a feasible point, we consider the case

$$\nabla_x f(x_F) = \nu \nabla_x h(x_F)$$

where ν is a scalar. It occurs if δx is orthogonal to $\nabla_x h(x_F)$, so that

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\delta x \cdot \big(-\nabla_x f(x_F)\big) = -\delta x \cdot \nu \nabla_x h(x_F) = 0$$

So a constrained local optimum occurs at x^* when $\nabla_x f(x^*)$ and $\nabla_x h(x^*)$ are parallel, that is,

$$\nabla_x f(x^*) = \nu \nabla_x h(x^*)$$

18.2.2.2 The inequality constraint

Again, x_F denotes a feasible point. The necessary and sufficient conditions for a constrained local minimum are the same as for an unconstrained local minimum. However, we observe two cases

1. the constraint is not active at the local minimum ($g(x^*) < 0$): therefore the local minimum is identified by the same conditions as in the unconstrained case.
2. the constrained local minimum occurs on the surface of the constraint surface: thus, we have an optimisation problem with an equality constraint ($g(x) = 0$). A local optimum occurs when $\nabla_x f(x)$ and $\nabla_x g(x)$ are parallel

   $$-\nabla_x f(x) = \lambda \nabla_x g(x)$$

   However, for some points, we do not have a local minimum since $-\nabla_x f(x_F)$ points in towards the feasible region. Thus, the constrained local minimum occurs when $-\nabla_x f(x)$ and $\Delta_x g(x)$ point in the same direction

   $$-\nabla_x f(x) = \lambda \nabla_x g(x) \text{ and } \lambda > 0$$

18.2.2.3 A summary

To summarise, if x^* corresponds to a constrained local minimum, then

1. unconstrained local minimum occurs in the feasible region
   (a) $g(x^*) < 0$
   (b) $\nabla_x f(x^*) = 0$
   (c) $\nabla_{xx} f(x^*)$ is a positive semi-definite matrix
2. unconstrained local minimum lies outside the feasible region
   (a) $g(x^*) = 0$
   (b) $-\nabla_x f(x^*) = \lambda \nabla_x g(x^*)$ with $\lambda > 0$
   (c) $v^\top \nabla_{xx} L(x^*) v \geqslant 0$ for all v orthogonal to $\nabla_x g(x^*)$

18.2.3 Conditions for a global optimum

Convexity plays a key role in mathematical programming. Convex programs minimise convex optimisation functions subject to convex constraints ensuring that every local minimum is always a global minimum. The second order convexity conditions are defined as follows:

Theorem 18.2.1 *Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable. Then f is convex if and only if for all $x \in \operatorname{dom} f$,*

$$\nabla^2 f(x) \geqslant 0$$

In general, we want the optimisation problem to be convex. We define a convex optimisation problem as follow:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 18.2.1 *An optimisation problem is convex if its objective is a convex function, the inequality constraint g is convex and the equality constraint is affine.*

$$\min_x f(x) \text{ (convex function) such that } g(x) \leqslant 0 \text{ (convex set) }, h(x) = 0 \text{ (affine)}$$

Theorem 18.2.2 *If $\widehat{x}$ is a local minimiser of a convex optimisation problem, it is a global minimiser.*

The following theorem emphasises the reasons why we want a convex problem. It applies to smooth functions.

Theorem 18.2.3 $\nabla f(x) = 0$ *if and only if x is a global minimiser of $f(x)$.*

Proof: For $\nabla f(x) = 0$ we have

$$f(y) \geqslant f(x) + \nabla f^\top(x)(y - x) = f(x)$$

For $\nabla f(x) \neq 0$ there is a direction of descent.

18.2.4 Some solutions

When dealing with constrained optimisation problem we want to transform the constraint problem to one of the following:

- a series of unconstrained problems
- a single but larger unconstrained problem
- another simpler constraint problem (dual, convex)

Some of the existing solutions are

1. Penalty and barriers
   - Associate a (adaptive) penalty cost with violation of the constraint
   - Associate an additional force compensating the gradient into the constraint (augmented Lagrangian)
   - Associate a log barrier with a constraint, becoming ∞ for violation (interior point method)
2. Gradient projection methods (mostly for linear constraints)
   - For active constraints, project the step direction to become tangantial
   - When checking a step, always pull it back to the feasible region
3. Lagrangian and dual methods
   - Rewrite the constrained problem into an unconstrained one
   - Or rewrite it as a (convex) dual problem
4. Simplex methods (linear constraints)
   - Walk along the constraint boundaries

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

18.3 Unconstrained optimisation

18.3.1 Some classical methods

18.3.1.1 The gradient descent

One of the simplest approach for solving convex unconstrained optimisation problem is the Gradient Descent (GD). The gradient descent search determines a vector x minimising the function $f(x)$ by starting with an arbitrary initial vector, and then repeatedly modifying it in small steps. At each step, the vector is altered in the direction that produces the steepest descent along the error surface, until a global minimum error is reached. This direction is found by computing the derivative of f with respect to each component of the vector x. It is called the gradient of f with respect to x, given by

$$\nabla f(x) = [\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n}]$$

***Remark* 18.3.1** *The (ascent) gradient specifies the direction that produces the steepest increase in f, while its negative produces the steepest decrease.*

The Gradient Descent is described in Algorithm (56).

Algorithm 56 Gradient Descent

```
for t = 1 to T do
    x_{t+1} ← x_t + Δx_t where Δx_t = −η_t ∇f(x_t)
end for
```

The update for the gradient descent is given by

$$x_{t+1} = x_t - \eta_t \nabla f(x_t)$$

In the case of the gradient ascent we get $\Delta x_t = \eta_t \nabla f(x_t)$ and the update is given by

$$x_{t+1} = x_t + \eta_t \nabla f(x_t)$$

In that setting t is the time step and η_t is the step size or learning rate. There are several ways for choosing the step size such as:

- Exact line search

$$\eta_t = \arg\min_{\eta} f\big(x - \eta \nabla f(x)\big)$$

- Backtracking line search. Let $\alpha \in (0, \frac{1}{2})$, $\beta \in (0, 1)$. Multiply $\eta = \beta\eta$ until

$$f\big(x - \eta \nabla f(x)\big) \leqslant f(x) - \alpha\eta \|\nabla f(x)\|^2$$

18.3.1.2 The subgradient descent

We can simplify the Gradient Descent method by using the Subgradient method (see Appendix (23.2.1.3)), which are iterative methods for solving convex minimisation problems (see Shor [1985]). They are convergent even when applied to a non-differentiable objective function. When the objective function is differentiable, sub-gradient methods for unconstrained problems use the same search direction as the method of steepest descent. See Algorithm (57).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 57 Subgradient Descent

```
for t = 1 to T do
    x_{t+1} ← x_t − η_t ψ_t
end for
```

where η_t is the step size and $\psi_t \in \partial f(x_t)$ denotes a subgradient of f at x_t and x_t is the t-th iterate of x. If f is differentiable, then its only subgradient is the gradient vector ∇f itself. It may happen that ψ_t is not a descent direction for f at x_t. We therefore maintain a list of f_{best} that keeps track of the lowest objective function value found so far. It is given by

$$f_{best,t} = \{f_{best,t-1}, f(x_t)\}$$

Many different types of step-size rules are used by subgradient methods. We consider methods with proof of convergence, where the step-sizes are determined off-line, before the method is iterated such as

- Constant step size: $\eta_t = \eta$
- Constant step length: $\eta_t = \frac{\gamma}{\|\psi_t\|_2}$ which gives $\|x_{t+1} - x_t\|_2 = \gamma$.
- Square summable but not summable step size, that is, any step sizes satisfying

$$\eta_t \geqslant 0\,,\ \sum_{t=1}^{\infty} \eta_t^2 \leqslant \infty\,,\ \sum_{t=1}^{\infty} \eta_t = \infty$$

- Nonsummable diminishing, that is, any step sizes satisfying

$$\eta_t \geqslant 0\,,\ \lim_{t\to\infty} \eta_t = 0\,,\ \sum_{t=1}^{\infty} \eta_t = \infty$$

Some examples of non-differentiable convex functions used in machine learning are

$$f(x) = \left(1 - a^\top x\right)^+\,,\ f(x) = \|x\|_1\,,\ f(x) = \sum_{r=1}^{k} \sigma_r(x)$$

where σ_r is the rth singular value of x.
Shor proved that for constant step-length and scaled subgradients having Euclidean norm equal to one, the subgradient method converges to an arbitrarily close approximation to the minimum value, that is,

$$\lim_{k\to\infty} f_{best,k} - f^* < \epsilon$$

18.3.1.3 The Newton's method

The idea behind the Newton's method is to use a second-order approximation to the function

$$f(x + \Delta x) \approx f(x) + \nabla f^\top(x)\Delta x + \frac{1}{2}\Delta x^\top \nabla^2 f(x)\Delta x$$

Choose Δx to minimise the above equation, we get

$$\Delta x = -\left(\nabla^2 f(x)\right)^{-1}\Delta f(x)$$

We get the descent direction

$$\nabla f^\top(x)\Delta x = -\nabla f^\top(x)\left(\nabla^2 f(x)\right)^{-1}\Delta f(x) < 0$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

18.3.1.4 The convergence

The convergence rate is as follows:

- Strongly convex case: $\nabla^2 f(x) \geqslant mI$, then we have Linear convergence. For some $\gamma \in (0,1)$, $f(x_t) - f(x^*) \leqslant \gamma^t$, $\gamma < 1$. We get
$$f(x_t) - f(x^*) \leqslant \gamma^t \text{ or } t \geqslant \frac{1}{\gamma} \log \frac{1}{\epsilon} \Rightarrow f(x_t) - f(x^*) \leqslant \epsilon$$
- Smooth case: $\|\nabla f(x) - \nabla f(y)\| \leqslant C\|x - y\|$. We get
$$f(x_t) - f(x^*) \leqslant \frac{K}{t^2}$$
- Newton's method is often faster, especially when f has long valleys.

Note, inverting a Hessian is very expensive $O(d^3)$. One can approximate the inverse Hessian with BFGS or Limited-memory BFGS. We can also use the Conjugate Gradient Descent. For unconstrained non-convex problems, theses methods will find local optima.

18.3.2 Adding constraints

18.3.2.1 The projected subgradient descent

One extension is the projected subgradient descent (PSD) which solves a constrained optimisation problem (see Lemarechal [2001]). Subgradient-projection methods are suitable for convex minimisation problems with very large number of dimensions, since they require little storage. Given a convex constraint set X we want to minimise $\min_{x \in X} f(x)$. We do subgradient steps and project x_t back into X at every iteration

$$x_{t+1} = \prod_X (x_t - \eta \psi_t)$$

where $\prod_X$ is a projection on X and ψ_t is any subgradient of f at x_t. Sketch of proof:

$$\| \prod_X (x_t) - x^* \| \leqslant \|x_t - x^*\|$$

if $x^* \in X$.
Any decreasing, non-summable step size $\eta_t \to 0$, $\sum_{t=1}^{\infty} \eta_t = \infty$ gives

$$f(x_{avg(t)}) - f(x^*) \to 0$$

Further, with $\eta_t \propto \frac{1}{\sqrt{t}}$, we get

$$f(x_{avg(t)}) - f(x^*) \leqslant \frac{C}{\sqrt{t}}$$

18.3.2.2 The subgradient descent

The subgradient method can be extended to solve the inequality constrained problem

$$\min f(x) \text{ such that } g_i(x) \leqslant 0 \text{ , } i = 1, ..., m$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $g_i(\cdot)$, $i = 1, ..., m$, are convex functions. The algorithm takes the same form as the unconstrained case in Algorithm (57). However, ψ_t is a subgradient of the objective or one of the constraint functions at x. That is,

$$\psi_t = \begin{cases} \partial f(x) \text{ if } g_i(x) \leqslant 0 \forall i = 1, ..., m \\ \partial g_j(x) \text{ for some } j \text{ such that } g_j(x) > 0 \end{cases}$$

where ∂f denotes the subdifferential of f. If the current point is feasible, the algorithm uses an objective subgradient, while if the current point is infeasible, the algorithm chooses a subgradient of any violated constraint.

18.3.2.3 The Newton method

The simpler case is Linear constraints $Ax = b$. For example, in the case of Newton method, we consider $f(x + \Delta x) - f(x)$ and minimise

$$\min_{\Delta x} \nabla f^\top(x)\Delta x + \frac{1}{2}\Delta x^\top \nabla^2 f(x)\Delta x \text{ such that } A\Delta x = 0$$

The solution Δx satisfies

$$A(x + \Delta x) = Ax + A\Delta x = b$$

In the case of inequality constraints we get

$$\min_{\Delta x} \nabla f^\top(x)\Delta x + \frac{1}{2}\Delta x^\top \nabla^2 f(x)\Delta x \text{ such that } g_i(x + \Delta x) \leqslant 0$$

18.4 A first approach to constrained optimisation

Since equality constraints can be expressed in terms of inequality constraints, we will focus on the latter.

18.4.1 Penalty and barriers

We use the following convention

- A barrier is ∞ for $g(x) > 0$
- A penalty is zero for $g(x) \leqslant 0$ and increases with $g(x) > 0$

We let $I(a) = 1$ if $a > 0$, 0 otherwise. We let $I_-(a) = \infty$ if $a > 0$, 0 otherwise. That is, $I_-(a) = \infty I(a)$. Further, we let $I_0(a) = \infty$ unless $a = 0$.
We present in Appendix (18.7) a generalisation of the penalty function approach. The penalty function has the general form

$$P(g(x)) = \Psi(g(x))$$

where the function $\Psi(x)$ needs to be defined. We are now going to discuss some of these functions.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

18.4.1.1 Log barrier method

In the logarithmic barrier method, rather than considering the problem in Equation (18.1.1), we rewrite the problem as

$$\min_x f(x) + \sum_{i=1}^m I_-(g_i(x))$$

where $I_-(a) = \infty$ if $a > 0$, 0 otherwise. That is, $I_-(a) = \infty I(a)$. Note, we get the approximation $I_-(a) \approx -\mu \log(-a)$ for small μ. Thus, we can rewrite the minimisation problem as

$$\min_x f(x) - \mu \sum_{i=1}^m \log(-g_i(x))$$

Some of the consequences of this re-formulation are:

- For $\mu \to 0$ then $-\mu \log(-g)$ converges to $\infty I(g > 0) = I_-(g > 0)$
- The barrier gradient $\nabla - \log(-g) = \frac{\nabla g}{g}$ pushes away from the constraint
- Eventually we want to have a very small μ, but doing so makes the barrier very non-smooth, which is bad for gradient and 2nd order methods

18.4.1.2 Central path

In this approach, every μ defines a different optimal $x^*(\mu)$ given by

$$x^*(\mu) = \arg\min_x f(x) - \mu \sum_{i=1}^m \log(-g_i(x))$$

As a result, each point on the path can be understood as the optimal compromise of minimising $f(x)$ and a repelling force of the constraints (which corresponds to the dual variables $\lambda^*(\mu)$). Detail is given in the Algorithm (58).

Algorithm 58 Log Barrier method

Require: Input: initial $x \in \mathbb{R}^n$, function $f(x)$, $g(x)$, $\nabla f(x)$, $\nabla g(x)$, tolerances θ, ϵ
Require: Output: x
Require: Initialise $\mu = 1$

```
repeat
    find x ← arg min_x f(x) − μ Σ_{i=1}^m log(−g_i(x)) with tolerance 10θ
    decrease μ ← μ/10
until |Δx| < θ and ∀i : g_i(x) < ε
```

18.4.1.3 Squared penalty method

In the Squared penalty method, rather than considering the problem in Equation (18.1.1), we rewrite the problem as

$$\min_x f(x) + \mu \sum_{i=1}^m I(g_i(x) > 0) g_i^2(x)$$

Note, this method will always lead to some violation of constraints. A better idea would be to add an out-pushing gradient/force $-\nabla g_i(x)$ for every constraint $g_i(x) > 0$ that is violated. Ideally, the out-pushing gradient mixes with $-\nabla f(x)$ exactly such that the result becomes tangential to the constraint. This idea leads to the augmented Lagrangian approach. Detail is given in the Algorithm (59).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 59 Squared Penalty method

Require: Input: initial $x \in \mathbb{R}^n$, function $f(x)$, $g(x)$, $\nabla f(x)$, $\nabla g(x)$, tolerances θ, ϵ
Require: Output: x
Require: Initialise $\mu = 1$

```
repeat
    find x ← arg min_x f(x) + μ Σ_{i=1}^m I(g_i(x) > 0) g_i^2(x) with tolerance 10θ
    decrease μ ← μ/10
until |Δx| < θ and ∀i : g_i(x) < ε
```

18.4.2 The augmented Lagrangian approach

18.4.2.1 The equality constraint

We first consider an equality constraint in Equation (18.1.1) before addressing inequalities which we rewrite as

$$\min_x f(x) + \mu \sum_{i=1}^m h_i^2(x) + \sum_{i=1}^m \lambda_i h_i(x)$$

Some of the consequences of this re-formulation are:

- The gradient $\nabla h_i(x)$ is always orthogonal to the constraint
- By tuning λ_i we can induce a virtual gradient $\lambda_i \nabla h_i(x)$
- The term $\mu \sum_{i=1}^m h_i^2(x)$ penalises the function as before

We first minimise the above equation for some μ and λ_i, which will lead to a (slight) penalty $\mu \sum_{i=1}^m h_i^2(x)$. For the next iteration, we choose λ_i to generate exactly the gradient that was previously generated by the penalty. The optimality condition after an iteration is

$$x^{'} = \arg\min_x f(x) + \mu \sum_{i=1}^m h_i^2(x) + \sum_{i=1}^m \lambda_i h_i(x)$$

which gives

$$0 = \nabla f(x^{'}) + \mu \sum_{i=1}^m 2h_i(x^{'}) \nabla h_i(x^{'}) + \sum_{i=1}^m \lambda_i \nabla h_i(x^{'})$$

We then update the λ for the next iteration

$$\begin{aligned} \sum_{i=1} \lambda_{i,new} \nabla h_i(x^{'}) &= \mu \sum_{i=1}^m 2h_i(x^{'}) \nabla h_i(x^{'}) + \sum_{i=1} \lambda_{i,old} \nabla h_i(x^{'}) \\ \lambda_{i,new} &= \lambda_{i,old} + 2\mu h_i(x^{'}) \end{aligned}$$

Detail is given in the Algorithm (60). As a consequence

- We do not have to take the penalty limit $\mu \to \infty$ but still can have exact constraints
- If f and h were linear (∇f and ∇h_i constant), the updated λ_i would be exactly right. That is, in the next iteration we would exactly hit the constraint (by construction)

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- The penalty term is like a measuring device for the necessary virtual gradient, which is generated by the augmentation term in the next iteration
- The λ_i are meaningful: they give the force/gradient that a constraint exerts on the solution

Algorithm 60 Augmented Lagrangian Equality method

Require: Input: initial $x \in \mathbb{R}^n$, function $f(x)$, $g(x)$, $\nabla f(x)$, $\nabla g(x)$, tolerances θ, ϵ
Require: Output: x
Require: Initialise $\mu = 1$, $\lambda_i = 0$
repeat
  find $x \leftarrow \arg\min_x f(x) + \mu \sum_{i=1}^m h_i^2(x) + \sum_{i=1}^m \lambda_i h_i(x)$
  $\forall i : \lambda_i \leftarrow \lambda_i + 2\mu h_i(x^{'})$
until $|\Delta x| < \theta$ and $|h_i(x)| < \epsilon$

18.4.2.2 The inequality constraint

We then consider the inequality constraint in Equation (18.1.1), which we rewrite as

$$\min_x f(x) + \mu \sum_{i=1}^m I\big(g_i(x) \geqslant 0 \vee \lambda_1(0)\big) g_i^2(x) + \sum_{i=1}^m \lambda_i g_i(x)$$

Note the λ_i are zero or positive but never negative. A constraint is either active or inactive:

- When active $\big(g_i(x) \geqslant 0 \vee \lambda_1(0)\big)$ we aim for equality $g_i(x) = 0$
- When inactive $\big(g_i(x) \geqslant 0 \wedge \lambda_1(0)\big)$ we do not penalise / augment equality $g_i(x) = 0$

Detail is given in the Algorithm (61).

Algorithm 61 Augmented Lagrangian Inequality method

Require: Input: initial $x \in \mathbb{R}^n$, function $f(x)$, $g(x)$, $\nabla f(x)$, $\nabla g(x)$, tolerances θ, ϵ
Require: Output: x
Initialise $\mu = 1$, $\lambda_i = 0$
repeat
  find $x \leftarrow \arg\min_x f(x) + \mu \sum_{i=1}^m I\big(g_i(x) \geqslant 0 \vee \lambda_1(0)\big) g_i^2(x) + \sum_{i=1}^m \lambda_i g_i(x)$
  $\forall i : \lambda_i \leftarrow \max\big(\lambda_i + 2\mu h_i(x^{'}), 0\big)$
until $|\Delta x| < \theta$ and $g_i(x) < \epsilon$

18.5 The Lagrangian approach

18.5.1 The KKT conditions

Given a constraint problem in Equation (18.1.1), we define the Lagrangian as

$$L(x, \lambda, \nu) = f(x) + \sum_{i=1}^m \lambda_i g_i(x) + \sum_{j=1}^l \nu_j h_j(x) \tag{18.5.3}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $\lambda_i \geqslant 0$, $\nu_j \in \mathbb{R}$ are called dual variables or Lagrange multipliers. We assume that f, g_i and h_j are convex and differentiable.
We will first focus on the case

$$L(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x)$$

The Lagrangian is useful when computing optima analytically. It implies the Karush-Kuhn-Tucker (KKT) conditions of optimality (see Kuhn et al. [1951]). That is, the KKT conditions encode the conditions for local minima described in Section (18.2.2). Note, the optima are necessarily at saddle points of the Lagrangian. The Lagrangian implies a dual problem, which is sometimes easier to solve than the primal one.
As discussed in Section (18.2.2), at the optimum there must be a balance between the cost gradient $-\nabla f(x)$ and the gradient of the active constraints $-\nabla g_i(x)$. Formally, for optimal $x : \nabla f(x) \in \mathrm{Span}\{\nabla g_i(x)\}$ or for optimal x, there must exist λ_i such that

$$-\nabla f(x) = \sum_{i=1}^{m} (-\lambda_i \nabla g_i(x))$$

Theorem 18.5.1 *KKT conditions*
Given the above Lagrangian, for optimal x, the Karush-Kuhn-Tucker conditions must hold (necessary condition): $\exists \lambda$ such that

$$\begin{aligned}
\nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x) &= 0 \textit{ force balance (or stationarity)} \\
\forall i : g_i(x) &\leqslant 0 \textit{ primal feasibility} \\
\forall i : \lambda_i &\geqslant 0 \textit{ dual feasibility} \\
\forall i : \lambda_i g_i(x) &= 0 \textit{ complementary slackness} \\
\textit{plus positive definite constraints on } \nabla_{xx} L(x, \lambda) &
\end{aligned}$$

The first condition can be equivalently expressed as $\exists \lambda$ such that

$$\nabla_x L(x, \lambda) = 0$$

The second condition corresponds to

$$\nabla_{\lambda_i} L(x, \lambda) = g_i(x) \leqslant 0 \,, \forall i$$

The fourth condition corresponds to

$$\sum_{i=1}^{m} \lambda_i g_i(x) = 0$$

The KKT conditions imply

1. Case 1: inactive constraint
   - when $\lambda^* = 0$ then we have $L(x^*, \lambda^*) = f(x^*)$
   - condition KKT [1] $\Rightarrow \nabla_x f(x^*) = 0$
   - condition KKT [2] $\Rightarrow x^*$ is a feasible point

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. case 2: active constraint
   - when $\lambda^* > 0$ then we have $L(x^*, \lambda^*) = f(x^*) + \lambda^* g(x^*)$
   - condition KKT [1] $\Rightarrow \nabla_x f(x^*) = -\lambda^* \nabla_x g(x^*)$
   - condition KKT [4] $\Rightarrow g(x^*) = 0$
   - condition KKT [4] also implies $\Rightarrow L(x^*, \lambda^*) = f(x^*)$

Thus, the Lagrangian can be viewed as the energy function that generates (for good choice of λ) the right balance between cost and constraint gradients. Note, this is like in the augmented Lagrangian approach, where however we have an additional augmented squared penalty that is used to tune the λ_i.
In the case of multiple equality and inequality constraints, the Karush-Kuhn-Tucker conditions must hold (necessary condition): $\exists \lambda$ such that

$$\begin{aligned}
\nabla_x L(x, \lambda, \nu) &= 0 \text{ force balance (or stationarity)} \\
\forall i : g_i(x) &\leqslant 0 \text{ primal feasibility} \\
\forall i : \lambda_i &\geqslant 0 \text{ dual feasibility} \\
\forall i : \lambda_i g_i(x) &= 0 \text{ complementary slackness} \\
h(x) &= 0 \text{ primal feasibility} \\
\text{plus positive definite constraints on } \nabla_{xx} L(x, \lambda) &
\end{aligned}$$

***Remark* 18.5.1** *Concerning the stationarity condition: for a differentiable function f, we can not use $\partial f(x) = \{\nabla f(x)\}$ unless f is convex.*

18.5.2 Implication from the KKT conditions

18.5.2.1 Equality constraints

In the case of the equality, the Lagrangian is given by

$$L(x, \nu) = f(x) + \sum_{j=1}^{l} \nu_j h_j(x) = f(x) + \nu^\top h(x)$$

Given the solution in Section (18.2.2), we get the following relations

- $\min_x L(x, \nu) \Rightarrow 0 = \nabla_x L(x, \nu) \longleftrightarrow$ force balance
- $\max_\nu L(x, \nu) \Rightarrow 0 = \nabla_\nu L(x, \nu) = h_i(x) \longleftrightarrow$ constraint

As a result, the optima (x^*, ν^*) are saddle points where

- $\nabla_x L = 0$ ensures force balance, and
- $\nabla_\nu L = 0$ ensures the constraint

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

18.5.2.2 Inequality constraints

In the case of the inequality, the Lagrangian is given by

$$L(x,\lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) = f(x) + \lambda^\top g(x)$$

and we get

$$\max_{\lambda \geqslant 0} L(x,\lambda) = \begin{cases} f(x) \text{ if } g(x) \leqslant 0 \\ \infty \text{ otherwise} \end{cases}$$

and

$$\max_{\lambda_i \geqslant 0} L(x,\lambda) = \begin{cases} \lambda_i = 0 \text{ if } g(x) < 0 \\ 0 = \nabla_{\lambda_i} L(x,\lambda) = g_i(0) \text{ otherwise} \end{cases}$$

It implies either $\big(\lambda_i = 0 \wedge g_i(x) < 0\big)$ or $g_i(0) = 0$, which is exactly equivalent to the KKT conditions. Again, the optima (x^*, λ^*) are saddle points where

- $\min_x L$ enforces force balance, and
- $\max_\lambda L$ enforces the KKT conditions

18.5.3 The Lagrangian dual problem

18.5.3.1 Definition

We define the Lagrange dual function as

$$l(\lambda,\nu) = \inf_x L(x,\lambda,\nu)$$

which implies two problems:

1. primal problem: $\min_x f(x)$ such that $g(x) \leqslant 0$ and $h(x) = 0$
2. dual problem: $\max_{\lambda,\nu} l(\lambda,\nu)$ such that $\lambda \geqslant 0$

Note, the dual problem is convex, even if the primal problem is non-convex. That is, $l(\cdot,\cdot)$ is concave. Thus, we can rewrite the two problems as

1. primal problem: $\min_x \big[\sup_{\lambda \geqslant 0,\nu} L(x,\lambda,\nu)\big]$
2. dual problem: $\max_{\lambda \geqslant 0,\nu} \big[\inf_x L(x,\lambda,\nu)\big]$

The primal problem is equivalent to a min-max optimisation because $\max_{\lambda \geqslant 0,\nu} L(x,\lambda,\nu)$ ensures the constraints. For example, consider a two-player game. If player 1 chooses x that violates a constraint $g_1(x) > 0$, player 2 choose $\lambda_1 \to \infty$ so that $L(x,\lambda,\nu) = \cdots + \lambda_1 f_1(x) + \cdots \to \infty$. Therefore, player 1 is forced to satisfy the constraints.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

18.5.3.2 Relation between primal and dual solutions

We now discuss the relation between primal and dual solutions in the case of inequality constraints. For any $\lambda_i \geqslant 0$, the dual function is always a lower bound

$$l(\lambda) = \min_x L(x, \lambda) \leqslant \min_x f(x) \text{ such that } g(x) \leqslant 0$$

and consequently

$$\max_{\lambda \geqslant 0}\big[\min_x L(x, \lambda)\big] \leqslant \min_x\big[\max_{\lambda \geqslant 0} L(x, \lambda)\big]$$

Including equality and inequality, we get:

Lemma 18.5.1 *Weak duality*
If $\lambda \geqslant 0$, then

$$l(\lambda, \nu) \leqslant f(x^*)$$

Proof: We have

$$\begin{aligned} l(\lambda, \nu) &= \inf_x L(x, \lambda, \nu) \leqslant L(x^*, \lambda, \nu) \\ &= f(x^*) + \sum_{i=1}^m \lambda_i g_i(x^*) + \sum_{j=1}^l \nu_j h_j(x^*) \leqslant f(x^*) \end{aligned}$$

Strong duality holds if and only if

$$\max_{\lambda \geqslant 0, \nu}\big[\min_x L(x, \lambda, \nu)\big] = \min_x\big[\max_{\lambda \geqslant 0, \nu} L(x, \lambda, \nu)\big]$$

If the primal is convex, and there exists an interior point, we get the Slater condition

$$\exists x : \forall i : g_i(x) < 0 \text{ and } \forall j : h_j(x) = 0$$

then we have strong duality. That is, primal and dual solutions are equivalent.

Theorem 18.5.2 *Strong duality*
For reasonable convex problems,

$$\sup_{\lambda \geqslant 0, \nu} l(\lambda, \nu) = f(x^*)$$

18.5.3.3 Interpretation

We can interpret duality as a linear approximation. We let $I_-(a) = \infty$ if $a > 0$ and 0 otherwise. We let $I_0(a) = \infty$ unless $a = 0$. We can rewrite the problem as

$$\min_x f(x) + \sum_{i=1}^m I_-(g_i(x)) + \sum_{j=1}^l I_0(h_j(x))$$

We replace $I_-(g_i(x))$ with $\lambda_i g_i(x)$, a measure of displeasure when $\lambda_i \geqslant 0$, $g_i(x) > 0$. Further, $\nu_j h_j(x)$ is a lower bound for $I_0(h_j(x))$. Thus, we get

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\min_x f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{l} \nu_j h_j(x)$$

As an example, we consider the linearly constrained least squares. The problem is as follows:

$$\min_x \frac{1}{2}\|Ax - b\|^2 \text{ such that } Bx = d$$

From the Lagrangian, we get

$$L(x, \nu) = \frac{1}{2}\|Ax - b\|^2 + \nu^\top (Bx - d)$$

Take the infimum, we get

$$\nabla_x L(x, \nu) = A^\top Ax - A^\top b + B^\top \nu \Rightarrow x = (A^\top A)^{-1}\big(A^\top b - B^\top \nu\big)$$

Replace the value of x and we get a simple unconstrained quadratic problem

$$\inf_x L(x, \nu) = \frac{1}{2}\|A\big[(A^\top A)^{-1}\big(A^\top b - B^\top \nu\big)\big] - b\|^2 + \nu^\top \Big(B\big[(A^\top A)^{-1}\big(A^\top b - B^\top \nu\big)\big] - d\Big)$$

18.5.3.4 Summary

When we have a constrained optimisation problem which is hard to solve, we might consider the dual problem since it may have simpler constraints. Further the solution of the latter is also the solution of the former.
Given the primal feasible x and the dual feasible λ and ν, the quantity

$$f(x) - l(\lambda, \nu)$$

is called the duality gap between x and λ, ν. We get a certificate of optimality: if we have a feasible x and know the dual $l(\lambda, \nu)$, then

$$\begin{aligned} l(\lambda, \nu) \leqslant f(x^*) \leqslant f(x) \quad &\Rightarrow \quad f(x^*) - f(x) \geqslant l(\lambda, \nu) - f(x) \\ &\Rightarrow \quad f(x) - f(x^*) \leqslant f(x) - l(\lambda, \nu) \end{aligned}$$

Thus, if the duality gap is zero, then x is primal optimal (and λ, ν are dual optimal). That is, all these inequalities are actually equalities. It provides a stopping criterion for a numerical implementation: if $f(x) - l(\lambda, \nu) \leqslant \epsilon$, then we are guaranteed that $f(x) - f(x^*) \leqslant \epsilon$.
We now introduce necessity and sufficiency for an optimum solution:

- Necessity states that if x^* and λ^*, ν^* are primal and dual solutions, with zero duality gap, then x^*, λ^*, ν^* satisfy the KKT conditions.
- Sufficiency states that if there exists x^*, λ^*, ν^* that satisfy the KKT conditions, then

$$\begin{aligned} l(\lambda^*, \nu^*) &= f(x^*) + \sum_{i=1}^{m} \lambda_i^* g_i(*) + \sum_{j=1}^{l} \nu_j^* h_j(*) \\ &= f(x^*) \end{aligned}$$

  where the first equality holds from stationarity, and the second holds from complementary slackness. Therefore, duality gap is zero so that x^* and λ^*, ν^* are primal and dual optimal.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In summary, KKT conditions are

- always sufficient
- necessary under strong duality

One of the most important uses of duality is that, under strong duality, we can characterise primal solutions from dual solutions. That is, for a problem with strong duality (assume Slater's condition) x^* and λ^*, ν^* are primal and dual solutions $\Longleftrightarrow$ x^* and λ^*, ν^* satisfy the KKT conditions.

If $\min_x L(x, \lambda, \nu)$ can be solved analytically, so does the convex dual problem. More generally, we have

$$\text{Optimisation problem} \;\rightarrow\; \text{Solve KKT conditions}$$

Thus, we can apply standard algorithms for solving an equation system $r(x, \lambda) = 0$ such as the Newton method

$$\nabla r \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = -r$$

It leads to primal-dual algorithms that adapt x and λ concurrently. They use the curvature $\nabla^2 f$ to estimate the right λ to push out of the constraint.

18.5.3.5 Equivalence

In general, one switch back and forth between the constrained form

$$\min_x f(x) \text{ such that } g(x) \leqslant d \text{ (C)}$$

where $d \in \mathbb{R}$ is a tuning parameter, to the Lagrange form

$$\min_x f(x) + \lambda \cdot g(x) \text{ (L)}$$

where $\lambda \geqslant 0$ is a tuning parameter, and claim the two methods are equivalent. We get two cases

1. (C) to (L): if the problem (C) is strictly feasible, then strong duality holds, and there exists some $\lambda \geqslant 0$ (dual solution) such that any solution x^* in (C) minimises

$$f(x) + \lambda \cdot \big(f(x) - d\big)$$

   so that x^* is also a solution in (L).

2. (L) to (C): if x^* is a solution in (L), then the KKT conditions for (C) are satisfied by taking $d = g(x^*)$ so that x^* is a solution in (C).

Thus, we get the equivalence

$$\begin{aligned} \bigcup_{\lambda \geqslant 0} \{ \text{ solutions in (L) } \} &\subseteq \bigcup_{d} \{ \text{ solutions in (C) } \} \\ \bigcup_{\lambda \geqslant 0} \{ \text{ solutions in (L) } \} &\supseteq \bigcup_{d \text{ such that (C) is strictly feasible}} \{ \text{ solutions in (C) } \} \end{aligned}$$

Note, this is not a perfect equivalence (albeit minor non-equivalence).
Using the KKT conditions and simple probability arguments, we get the following result:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 18.5.3 *Let f be differentiable and strictly convex, $A \in \mathbb{R}^{n\times p}$, $\lambda > 0$. Consider*

$$\min_x f(Ax) + \lambda\|x\|_1$$

If the entries of A are drawn from a continuous probability distribution (on $\mathbb{R}^{n\times p}$), then with probability 1, the solution $x^* \in \mathbb{R}^p$ is unique and has at most $\min\{n, p\}$ nonzero components.

Note, while f must be strictly convex, there is no restrictions on the dimensions of A and one could choose $p >> n$.

18.5.4 Additional

18.5.4.1 The log barrier method revisited

Going back to the log barrier method in Section (18.4.1.1), for a given μ, the optimality condition is

$$\nabla f(x) - \sum_{i=1}^m \frac{\mu}{g_i(x)}\nabla g_i(x) = 0$$

or equivalently, we get the modified (approximated) KKT conditions

$$\nabla f(x) - \sum_{i=1}^m \lambda_i \nabla g_i(x) = 0 \,,\, \lambda_i g_i(x) = -\mu$$

We see that centring in the log barrier method is equivalent to solving the modified KKT conditions. In addition, on the central path, the duality gap is μ:

$$l(\lambda^*(\mu)) = f(x^*(\mu)) + \sum_i \lambda_i g_i(x^*(\mu)) = f(x^*(\mu)) - \mu$$

18.5.4.2 Non-convex optimisation

We consider the optimisation problem with equality constraint

$$\min_x f(x) \text{ such that } h(x) = 0$$

where both $f(\cdot)$ and $h(\cdot)$ are neither convex nor linear. Given the Lagrangian

$$L(x, \nu) = f(x) + \nu^\top h(x)$$

we get the min-max problem

$$\min_x \max_\nu L(x, \nu)$$

The idea is to let x_0 be an initial value of x and compute the increment dx and the value of ν by solving a linearised version of the above problem, replacing x by $x + dx$, and iterates. If the function $f(\cdot)$ is differentiable, the increment dx is normally computed at each iteration by solving the linear system

$$\begin{pmatrix} \eta I & \nabla_x h^\top \\ \nabla_x h & 0 \end{pmatrix} \begin{pmatrix} dx \\ \nu \end{pmatrix} = \begin{pmatrix} -\nabla_x f \\ -h(x) \end{pmatrix}$$

which amounts to projected gradient descent. The projection is performed onto the hyperplanes of linearised constraints. This system is derived from the KKT conditions that dx and ν must satisfy.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, when the function $f(\cdot)$ is a sum of squared residuals (typical in regression problems), one can replace the projected gradient by Gauss-Newton (GN) or Levenberg-Marquardt (LM) (see Fua et al. [2010]). The iteration scheme remains the same but the KTT conditions become

$$\begin{pmatrix} J^\top J + \eta I & \nabla_x h^\top \\ \nabla_x h & 0 \end{pmatrix} \begin{pmatrix} dx \\ \nu \end{pmatrix} = \begin{pmatrix} -J^\top r(x_t) \\ -h(x_t) \end{pmatrix}$$

where r is the vector of residuals and $J = \nabla_x r$ is its Jacobian matrix evaluated at x_t.

18.6 Standard convex programs

There exist a large variety of mathematical programs, each being a different research area in itself with extensive theory and algorithms. Given the constraint problem in Equation (18.1.1), we review a few basic convex optimisation programming models and present some examples. See Nocedal et al. [1999] for details. See Press et al [1992] for numerical implementation.

18.6.1 Regularisation

A linear operator problem consists in finding $x \in X$ satisfying $Ax = b$, where A is a linear operator from a normed space X to a normed space Y, and $b \in Y$ is a predetermined constant. The problem is ill-posed if small deviations in b, due to noise, result in large deviations in the corresponding solutions. That is, if b changes to b^δ, where $\|b^\delta - b\| < \delta$, then finding x minimising the functional $R(x) = \|Ax - b^\delta\|^2$ does not guarantee a good approximation to the desired solution even if δ tends to zero.
Ivanov [1962] and Tikhonov [1963] showed that rather than minimising $R(x)$, we should minimise the regularised functional

$$\widetilde{R}(x) = \|Ax - b^\delta\|_2^2 + \gamma(\delta)P(x)$$

where $P(x)$ is some functional and $\gamma(\delta)$ is an appropriately chosen constant. In that case, we obtain a sequence of solutions that does converge to the desired one as δ tends to zero. This led to the theory of regularisation.
It has been used in the field of machine learning, especially in the literature on high-dimensional regression (see Candes et al. [2007]), where one want to solve

$$\min_{\beta \in \mathbb{R}^p} \|y - X\beta\|_2 + \lambda P(\beta)$$

where $P(\beta) = \|\beta\|_1$, $y = [y_1, ..., y_n] \in \mathbb{R}^n$ is the observed data, $X = [X_1, ..., X_n] \in \mathbb{R}^{n \times p}$ is the vector of covariates, $\beta \in \mathbb{R}^p$ is the regression coefficient best fitting the linear model $y = X\beta$, and $\lambda > 0$ is a parameter controlling the sparsity of the solution. This is the Lasso model, used in high-dimensional applications where one requires sparsity of the solution β for interpretability and recovery purpose when p is large. In the case of the Tikhonov regularisation function $P(\beta) = \|\beta\|_2$, we can deal with issues arising when the data matrix X is ill-conditioned of singular.

18.6.2 Simple programming models

18.6.2.1 Quadratic programming

Quadratic programming is used in problems minimising least squares loss function. A quadratic program (QP) has a quadratic objective with linear constraints. The problem is stated as

$$\min_x \frac{1}{2} x^\top Q x + C^\top x \text{ such that } a_i x \leqslant b_i \text{ , } i \in I \text{ , } a_j x = b_j \text{ , } j \in \varepsilon$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where the Hessian matrix Q is $n \times n$ symmetric, I and ε are finite sets of indices and a_i, $i \in I \bigcup \varepsilon$ are $n \times 1$ vectors. As discussed above, if the matrix Q is positive, $x^\top Qx \geqslant 0$ for any x, then the problem is convex. For convex QP, any local solution is also a global solution. A QP can always be solved or shown to be infeasible in a finite number of iterations. AS discussed in Section (18.5), the KKT optimal conditions are

1. $Qx + \sum_{i \in I} \lambda_i a_i + \sum_{j \in \varepsilon} \nu_j a_j = 0$ Dual Feasibility
2. $a_i^\top x \leqslant b_i$ for $i \in I$ Primal Feasibility
3. $a_j^\top x = b_j$ for $j \in \varepsilon$ Primal Feasibility
4. $\lambda_i \left(a_i^\top x - b_i\right) = 0$ for $i \in I$ Complementary

In the special case where there are no constraints ($I = 0$), then the KKT point can be found by simply solving a system of linear equations. It is more difficult in presence of inequality constraints. Two methods exists

1. interior point methods
2. active-set methods: the optimal active set is the set of constraints satisfied as equalities at the optimal solutions. Active set methods work by making educated guesses as to the active set and solving the resulting equality constrained QP. In the case of wrong guess, one can use gradient and Lagrangian multiplier information to determine constraints to add to or subtract from the active set.

18.6.2.2 Linear programming

18.6.2.2.1 Linear programming In some practical optimisation problems we are interested in minimising costs or maximising profit values, which are often the sum of several independent weighted contributions. That is, linear functions. The conditions on feasible solutions can be formulated as linear inequalities or equations. It suggests the following problem classes: Linear programming optimises a linear function subject to linear constraints. That is, the set $\mathcal{S}$ is described by linear equations or inequalities. Since linear functions and constraints are convex, a linear program (LP) is always a convex program. Linear programming is a special case of the QP above, with the Hessian Q equal to zero. The problem is stated as

$$\min_x C^\top x \text{ such that } a_i x \leqslant b_i \text{ , } i \in I \text{ , } a_j x = b_j \text{ , } j \in \varepsilon$$

In general, LP can be efficiently solved by using interior point methods (see Khachiyan), or simplex methods (active set methods) (see Dantzig). Even though the simplex algorithm has an exponential worst case complexity, it runs fairly quickly in practice. Powerful computer programs for solving LPs exist, even when huge numbers of variables and inequalities are involved. Some examples are CPLEX and SoPlex which is part of the ZIB Optimization Suite.

18.6.2.2.2 Integer linear programming In linear programming the coordinates of solution vectors x are arbitrary real numbers. In the case where x is a vector with integer coordinates the problem is called an Integer Linear program (ILP). If x is further restricted to values 0 and 1, the ILP is called a Binary Integer program (BIP). Solving such problems is NP-hard. At last, the Mixed Integer Linear program (MIP) has the general form

$$\max c^\top x + h^\top y \text{ such that } Ax + Gy \leqslant b \text{ , } x, y \geqslant 0$$

where x and y is a vector of integer and real variables, respectively. Thus, both LP and ILP are special cases of MIP.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

18.6.2.3 Second-order cone programming

The second-order cone programming (SOCP) problems have a linear objective, second-order cone constraints, and possibly additional linear constraints. The problem is stated as

$$\min_x C^\top x \text{ such that } \|R_i x + d_i\|_2 \leqslant a_i x + b_i \text{ , } i \in I \text{ , } a_j x = b_j \text{ , } j \in \varepsilon$$

where $R_i \in \mathbb{R}^{n_i \times n}$ and $d_i \in \mathbb{R}^{n_i}$. These problems are often solved by using interior point algorithms.

18.6.2.4 Semidefinite programming

Semidefinite programs (SDP) are the generalisation of linear programs (LP) to matrices. In standard form, a SDP minimises a linear function of a matrix subject to linear equality constraints and a matrix non-negativity constraint. The problem is stated as

$$\min_X < C, X > \text{ such that } < A_i, X >= b_i \text{ , } i \in I \text{ , } X \succeq 0$$

where X, C and A_i take values in $\mathbb{R}^{n \times n}$ and $b_i \in \mathbb{R}$. Further, $X \succeq 0$ means X must be positive semidefinite and $< C, X >= \text{trace}(CX)$. In general, SDP are solved via interior programming methods (see Mittelmann [2003]).

18.6.3 Some examples

18.6.3.1 Quadratic programming

As an example we consider a quadratic function with equality constraints. Given $Q \geqslant 0$, the optimisation problem is

$$\min_x \frac{1}{2} x^\top Q x + c^\top x \text{ such that } Ax = 0$$

It is a convex problem with no inequality constraints so that by the KKT conditions x is a solution if and only if

$$\begin{pmatrix} Q & A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} -c \\ 0 \end{pmatrix}$$

for some λ. Linear system combines stationarity and primal feasibility (complementary slackness and dual feasibility are vacuous).

18.6.3.2 The Lasso problem

Another example is the Lasso problem (see Section (18.6.1)): given the response $y \in \mathbb{R}^n$, the predictors $A \in \mathbb{R}^{n \times p}$ (columns $A_1, ..., A_p$), the optimisation problem is

$$\min_{x \in \mathbb{R}^p} \frac{1}{2} \|y - Ax\|^2 + \nu \|x\|_1$$

The KKT conditions are

$$A^\top (y - Ax) = \lambda s$$

where $s \in \partial \|x\|_1$, that is,

$$s_i \in \begin{cases} \{1\} \text{ if } x_i > 0 \\ \{-1\} \text{ if } x_i < 0 \\ (-1, 1] \text{ if } x_i = 0 \end{cases}$$

Note, if $|A_i^\top (y - Ax)| < \nu$ then $x_i = 0$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

18.6.3.3 ILP formulations

We present some formulation of discrete problems introduced in Section (18.1.3).

18.6.3.3.1 $(0,1)$-Knapsack Since a solution is described by the information which items are in or out, we introduce $0,1$-variables x_j by

$$x_j = \begin{cases} 1 \text{ if item j is in the knapsack} \\ 0 \text{ otherwise} \end{cases}$$

Then the objective function is $\max \sum_{j=1}^{N} c_j x_j$, and the constraint $\sum_{j=1}^{N} a_j x_j \leqslant b$ ensures that the capacity is not exceeded.

18.6.3.3.2 Assignment problem We assume we are given N tasks and N machines capable of performing any of the tasks. We let c_{ij} be the cost of performing the task j with machine i. The decision variables are

$$x_{ij} = \begin{cases} 1 \text{ if machine i performs task j} \\ 0 \text{ otherwise} \end{cases}$$

Then the objective function is $\min \sum_{i,j=1}^{N} c_{ij} x_{ij}$ with the constraints

$$\forall j \in V : \sum_{i \in V} x_{ij} = 1 \text{ and } \forall i \in V : \sum_{j \in V} x_{ij} = 1$$

18.6.3.3.3 ATSP We denote the N nodes by $1, ..., N$. We let c_{ij} be the distance from vertex i to vertex j for $i, j = 1, ..., N$. We observe that a tour (cycle) is characterised by the set of used arcs, so that we can work with simple binary variables. We define the variables by

$$x_{ij} = \begin{cases} 1 \text{ if the tour uses } arc(i,j) \\ 0 \text{ otherwise} \end{cases}$$

The objective is $\min \sum_{ij} c_{ij} x_{ij}$ and the constraints are

$$\forall j \in V : \sum_{i \in V} x_{ij} = 1 \text{ (every vertex has one predecessor) and } \forall i \in V : \sum_{j \in V} x_{ij} = 1 \text{ (every vertex has one successor)}$$

which express that exactly one arc of the tour must enter and leave every node. These are called two-matching constraints. However, these $2N$ constraints are not enough since any union of disjoint directed cycles would also be a feasible solution. To rule out these short constraints, we need non-local constraints called subtour elimination constraints. Since a cycle has as many arcs as nodes, we can require that, in any proper subset $W \subset V$, fewer than $|W|$ arcs are used. This would destroy these unwanted short cycles. Thus, we create for all $W \subset V$ with $2 \leqslant |W| \leqslant |V| - 2$ the constraint

$$\sum_{(i,j) \in A(W)} x_{ij} \leqslant |W| - 1$$

where $A(W)$ denotes the set of arcs that have both endnodes in W. However, the number of subsets W is exponential in N, which is impracticable.
In the case of STSP, we only need one variable x_e for every undirected edge, and the constraints are

$$\forall v \in V : \sum_{e \in \delta(v)} x_e = 2 \text{ and } \sum_{e \in E(W)} x_e \leqslant |W| - 1$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

for all $W \subset V$ with $3 \leqslant |W| \leqslant |V| - 3$. Here $E(W)$ is the set of edges in W, and $\delta(v)$ is the set of edges with v as one endnode.

18.7 Generalising the penalty function approach

18.7.1 The energy function

As already seen above, the penalty function method consists of transforming the constrained optimisation problem in Equation (18.1.1) (but with $g_i(x) \geqslant 0$) into an unconstrained one based on a penalty function (see Luenberger [1984]). That is, the penalty function $E(\cdot, k)$ satisfies

$$E(x,k) = \alpha f(x) + \sum_{i=1}^{m} k_i P(g_i(x))$$

where $\alpha = +1$ for minimisation problems and $\alpha = -1$ for maximisation problems. The penalty term $P(g_i(x))$ should satisfy

$$\begin{aligned} P(g_i(x)) &= 0 \text{ if } g_i(x) \geqslant 0 \text{ the corresponding constraint is satisfied} \\ P(g_i(x)) &> 0 \text{ if } g_i(x) < 0 \end{aligned}$$

The constants k_i, $i = 1, ..., m$, define the relative weight concerning the satisfaction of some constraints against some other and/or the relative weight of satisfying all the constraints or minimising $sf(x)$. The task of tuning these parameters provides a means of customising the problem according to the current needs.
As discussed in Section (5.1.2), the function $E(\cdot, k)$ is also called the energy function for the corresponding network (see Cichocki et al. [1993]).
Assuming differentiability, a local minimum of the penalty function is obtained by using the dynamic gradient scheme

$$\frac{dx}{dt} = -\mu \nabla_x E(x,k) \, , x(0) = x^{(0)}$$

where t is the time step and

$$\mu = \text{diag}\big(\mu_1, \mu_2, ..., \mu_n\big)$$

or in extended form

$$\begin{aligned} \frac{dx_1}{dt} &= -\mu_1 \Big(\frac{\partial f(x)}{\partial x_1} + \sum_{i=1}^{m} k_i \frac{\partial P}{\partial g_i} \frac{\partial g_i(x)}{\partial x_1} \Big) \, , x_1(0) = x_1^{(0)} \\ \frac{dx_2}{dt} &= -\mu_2 \Big(\frac{\partial f(x)}{\partial x_2} + \sum_{i=1}^{m} k_i \frac{\partial P}{\partial g_i} \frac{\partial g_i(x)}{\partial x_2} \Big) \, , x_2(0) = x_2^{(0)} \\ &\vdots \\ \frac{dx_n}{dt} &= -\mu_n \Big(\frac{\partial f(x)}{\partial x_n} + \sum_{i=1}^{m} k_i \frac{\partial P}{\partial g_i} \frac{\partial g_i(x)}{\partial x_n} \Big) \, , x_n(0) = x_n^{(0)} \end{aligned}$$

where $\mu_j > 0$ and $k_i > 0$. Usually one takes $\mu_j = \mu = \frac{1}{\tau}$, $j = 1, ..., m$, where τ is a time constant, and $k_i = k$, $i = 1, ..., n$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

18.7.2 The penalty functions

The penalty function has the general form

$$P(g(x)) = \Psi(g(x))$$

where the function $\Psi(x)$ should penalise only configurations where $x > 0$. One possibility is to use a sigmoid function or

$$\Psi(x) = x\Theta(x)$$

where $\Theta(x)$ is a Heaviside function (see Ohlsson [1993]). The penalty term is zero if and only if the constraint is satisfied, otherwise the penalty is proportional (linear) to the degree of violation. The slope of $\Psi(x)$ is implicitly given by the strength of the constraint k in the energy function.
Some examples of penalty terms are

$$\begin{aligned} P(g_i(x)) &= \big[\min(0, g_i(x))\big]^2 \\ P(g_i(x)) &= -\min(0, g_i(x)) \end{aligned}$$

18.7.2.1 Type I

In the first case, the energy function becomes

$$E(x,k) = f(x) + \frac{1}{2}\sum_{i=1}^{m} k_i\big[\min(0, g_i(x))\big]^2$$

Assuming that $g_i(x)$, $i = 1, ..., m$, have continuous first derivatives, one can show that the same is true for $P(g_i(x))$, so that the energy function is continuously differentiable. The gradient of $P(g_i(x))$ is

$$\nabla_x\big[\min(0, g_i(x))\big]^2 = \min(0, g_i(x))\nabla_x g_i(x)$$

Using the gradient strategy for minimising the energy function $E(\cdot, k)$, we get the system of ordinary differential equations

$$\frac{dx_j}{dt} = -\mu_j\Big(\frac{\partial f(x)}{\partial x_j} + \sum_{i=1}^{m} k_i S_i g_i(x)\frac{\partial g_i(x)}{\partial x_j}\Big), j = 1, ..., n$$

where the control signals are

$$S_i = \begin{cases} 1 \text{ if } g_i(x) \leqslant 0 \\ 0 \text{ if } g_i(x) > 0 \end{cases}$$

and

$$S_i g_i(x) = \min(0, g_i(x))$$

Kennedy et al. [1988] and Cichocki et al. [1993] showed that the functional scheme for simulating these equations could be considered as a neural network (NN), where the integrators represent the neurons (basic units) and functional nonlinear generators build up the connections between them. This approach replaces the constrained problem in Equation (18.1.1) by an unconstrained minimisation of the differentiable penalty function $E(\cdot, k)$. Luenberger [1984] explained that theoretical results on the penalty function method show that equivalence of the problems in Equation (18.1.1) and $\min_x E(x, k)$ is only obtained in the limit, as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\min_{i=1,...,m} \{k_i\} \to \infty$$

Since the values of the parameters k_i, $i = 1, ..., m$, used in a NN are finite, it follows that the unconstrained minima of the energy function obtained by the NN will only be approximations to the true solutions of the constrained problem.

18.7.2.2 Type II

We now consider the case of the second penalty term, the energy function becomes

$$E(x,k) = f(x) - \sum_{i=1}^{m} k_i \min(0, g_i(x))$$

Due to the non-differentiability of the penalty term, this penalty function is non-differentiable, even though the functions $g_i(x)$ are assumed differentiable. Luenberger [1984] proved that any unconstrained minimum of the above penalty function is also a solution of the constrained problem in Equation (18.1.1) provided that $\min_{i=1,...,m}\{k_i\}$ is sufficiently large. Thus, an unconstrained minimisation will yield the true solution if k_i, $i = 1, ..., m$ are sufficiently large but finite. This solution is termed exact. In that case, the corresponding system of ordinary differential equations is

$$\frac{dx_j}{dt} = -\mu_j \Big(\frac{\partial f(x)}{\partial x_j} - \sum_{i=1}^{m} k_i S_i \frac{\partial g_i(x)}{\partial x_j} \Big) \, , j = 1, ..., n$$

where $\mu_j > 0$ and the control signals are

$$S_i = \begin{cases} 1 \text{ if } g_i(x) \leqslant 0 \\ 0 \text{ if } g_i(x) > 0 \end{cases}$$

The functional scheme is a simpler NN than with the previous penalty function using simple switches instead of the expensive analog multipliers. However due the non-smooth nature of the penalty function, the first derivative discontinuities can lead to parasitic effects slowing up the solution speed.

18.7.2.3 Type III

Rather than adding together the penalty terms $-k_i \min(0, g_i(x))$, Mladenov et al. [1999] proposed to take the maximum of these terms, getting the penalty (energy) function

$$\begin{aligned} E(x,k) &= f(x) + \max_{i=1,...,m} \big(k_i P(g_i(x))\big) \\ &= f(x) + \max_{i=1,...,m} \big(\max\big(0, -k_i g_i(x)\big)\big) \\ &= f(x) + \max\big(0, -k_1 g_1(x), -k_2 g_2(x), ..., -k_m g_1(m)\big) \end{aligned}$$

In that setting, only the most violated inequality constraint is taken into account. Note, this system is non-differentiable. Again, it has been shown that it is an exact penalty function (see Polak [1997]).
To apply the gradient strategy, we need to select

$$\frac{\partial x}{\partial t} \in -\mu \partial E(x,k)$$

where $\partial E(x,k)$ is the generalised gradient of the penalty function $E(\cdot,k)$. We use the same system of ordinary differential equations as in Type II except that the control signals S_i are defined as follows:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$S_i = \begin{cases} 1 \text{ if } \max\big(0, -g_i(x)\big) = \max_{j=1,\ldots,m}\Big(\max\big(0, -g_i(x)\big)\Big) \\ 0 \text{ if } \max\big(0, -g_i(x)\big) < \max_{j=1,\ldots,m}\Big(\max\big(0, -g_i(x)\big)\Big) \end{cases}$$

The proposed NN architecture consists in a winner-take-all block included into the Control Network (see Cichocki et al. [1993]). This block is used to determine dynamically the index i^* of the most violated inequality and to generate the control signal $S_{i*} = 1$ corresponding to this inequality, while all other control signals (corresponding to other inequalities) are equal to 0. The same non-smooth character of the exact penalty function as in Type II applies.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 19

Global search optimisation

As discussed in Section (1.1), among the three ways of achieving machine intelligence (see Turing [1948]) there is the genetic or evolutionary search, where a combination of genes is looked for and the success criterion is the survival value. Later, Turing [1950] proposed a learning machine which would parallel the principles of evolution. He stated that we could not find a good child-machine at the first attempt and that it should be thought in an evolutionary process. Computer simulation of evolution started with the work of Nils Aall Barricelli [1954] in Princeton. Starting in 1957, Alex Fraser published a series of papers on simulation of artificial selection of organisms with multiple loci controlling a measurable trait (see Fraser [1957]). Computer simulation of evolution was extended in the 1960s by biologists such as Fraser, Burnell, Crosby, and Bremermann, among others. Artificial evolution became a widely recognised optimisation method as a result of the work of Rechenberg and Schwefel in the 1960s and early 1970s. They were capable of solving complex engineering problems through evolution strategies. L.J. Fogel proposed the evolutionary programming technique for generating artificial intelligence. It used finite state machines for predicting environments, and used variation and selection to optimise the predictive logics. In particular, genetic algorithms (GA) became popular through the work of John Holland in the early 1970s. He introduced a formalised framework for predicting the quality of the next generation, known as Holland's Schema Theorem (see Holland [1973] [1975]). Further, the idea of evolving programs was current amongst John Holland's students. Cramer [1985] published evolved programs in two specially designed languages, and John Koza [1990] patented his invention of a GA for program evolution, which was followed by a publication (see Koza [1989]).
We are briefly going to introduce evolutionary algorithms, and present some algorithms, before discussing in greater depth Differential Evolution. We will then present some techniques for handling the constraints.

19.1 Evolutionary algorithms

19.1.1 Description

19.1.1.1 A brief history of evolutionary algorithms

Evolutionary algorithms (EAs) introduced by Holland [1962] [1975] and Fogel [1966] are robust and efficient optimisation algorithms based on the theory of evolution proposed by Darwin [1882], where a biological population evolves over generations to adapt to an environment by mutation, recombination and selection. They are stochastic search algorithms, searching from multiple points in space instead of moving from a single point like gradient-based methods do (see Menon [2004]). These algorithms are typically initiated with a population of potential solutions, that may be drawn randomly or specified prior to the beginning of the search. Iteration on the population is based on the principles of natural selection, with each iteration, or generation, improving in fitness as defined by some predetermined measure. The methods by which each generation is determined are specific to the particular algorithm in question, however, all evolutionary algorithms rely on classes of stochastic operator known as selection and re-

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

production operators. The selection operator acts to ensure that individuals with greater fitness in each generation are selected as parents for the next generation, whilst the reproduction operator determines how the next generation is derived from the parents selected. Moreover, they work on function evaluation alone (fitness) and do not require derivatives or gradients of the objective functions. McKay [2008] claimed that despite their simplicity, given modest resources and a relatively tough optimisation problem, Evolutionary Algorithms reliably converge to good solutions, and what's more, they are suited to parallel implementation due to their speed scaling almost linearly with the number of processors.

There is a large literature describing different evolutionary algorithms (EAs) commonly used to solve constrained non-linear programming problems (CNOPs) such as evolutionary programming, evolution strategies, genetic algorithms (GAs), differential evolution (DE) and many more. For instance, GAs are general purpose search algorithms based on an evolutionary paradigm where the population members are represented by strings, corresponding to chromosomes. Search starts with a population of randomly selected strings, and, from these, the next generation is created by using genetic operators (mutation). At each iteration individual strings are evaluated with respect to a performance criteria and assigned a fitness value. Strings are randomly selected using these fitness values to either survive or to mate to produce children for the next generation. However, among the different EA's commonly used DE became very popular. DE is a population-based approach to function optimisation generating a new position for an individual by calculating vector differences between other randomly selected members of the population. The DE algorithm is found to be a powerful evolutionary algorithm for global optimisation in many real problems. As a result, since the original article of Storn and Price [1995] many authors improved the DE model to increase the exploration and exploitation capabilities of the DE algorithm when solving optimisation problems.

Since EAs are search engines working in unconstrained search spaces they lacked until recently of a mechanism to deal with the constraints of the problems. The first attempts to handle the constraints were either to incorporate methods from mathematical programming algorithms within EAs such as penalty functions, or, to exploit the mathematical structure of the constraints. Then, a considerable amount of research proposed alternative methods to improve the search of the feasible global optimum solution. Most of the research on DE focused on solving CNOPs by using a sole DE variant, a combination of variants or combining DE with another search method. One of the most popular constraint handling mechanisms is the use of the three feasibility rules proposed by Deb [2000] on genetic algorithms. Using some of the improvements to the DE algorithm combined with simple and robust constraint handling mechanisms we propose a modified algorithm for solving our optimisation problem under constraints which greatly improves its performances.

19.1.1.2 Some terminology

Deoxyribonucleic acid (DNA) is a molecule composed of two chains that coil around each other to form a double helix carrying the genetic instructions used in the growth, development, functioning, and reproduction of all known living organisms and many viruses. DNA and ribonucleic acid (RNA) are nucleic acids.
The vast majority of organisms encode their genes in long strands of DNA. A gene is a sequence of nucleotides in DNA or RNA that codes for a molecule that has a function. During gene expression, the DNA is first copied into RNA. The RNA can be directly functional or be the intermediate template for a protein that performs a function. The transmission of genes to an organism's offspring is the basis of the inheritance of phenotypic trait. These genes make up different DNA sequences called genotypes. Genotypes along with environmental and developmental factors determine what the phenotypes will be. Most biological traits are under the influence of polygenes (many different genes) as well as gene-environment interactions. Some genetic traits are instantly visible, such as eye color or number of limbs, and some are not, such as blood type, risk for specific diseases, or the thousands of basic biochemical processes that constitute life.
Genes can acquire mutations in their sequence, leading to different variants, known as alleles, in the population. These alleles encode slightly different versions of a protein, which cause different phenotypical traits.
The total complement of genes in an organism or cell is known as its genome, which may be stored on one or more

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

chromosomes. A chromosome consists of a single, very long DNA helix on which thousands of genes are encoded. The region of the chromosome at which a particular gene is located is called its locus. Each locus contains one allele of a gene. However, members of a population may have different alleles at the locus, each with a slightly different gene sequence.

19.1.1.3 Adaptation to the environment

Natural evolution led to the development of complex organisms (such as plants and animals) from simpler single celled life forms. Adaptations are ways that animals have adjusted to be better suited for their environment. Physical adaptations are parts of the organism, such as ears or beaks. Behavioural adaptations, on the other hand, are actions that the organism takes that help it survive.
Rabbits have been described as animals that are cunning and able to come out of difficult situations rapidly. They have well known capacity for adaptations that have helped them survive through centuries and in different parts of the world from mountains to deserts, and plains to arctic areas, in spite of being prey animals. Each adaptation, like the teeth, the long ears, and the rapid reproduction rate are an essential survival tool that has kept the rabbit population of the world abundant and safe from predators. Some of the rabbits characteristics are:

- long ears
- eyes
- legs
- teeth
- digestion
- fur colour
- reproduction

In order to describe adaptation to the environment we consider two rabbits, a black and white BW one, and a ginger and white GW one. Their problem consists in surviving within their environment. See Figure (19.1).

Figure 19.1: Surviving within the environment

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The growth of animals is primarily controlled by the genes they inherited from their parents. The genes are stored on one or more strands of DNA. Both rabbits have different genes with different characteristics. See Figure (19.2).

Figure 19.2: Genes representation

In our example there is nine characteristics: both rabbits have four legs, and different colours. They differentiate themselves in the fact that BW has an $IQ = 3$ while GW has an $IQ = 5$. See Figure (19.3).

Figure 19.3: Genes characteristics

Natural evolution arises as only the fittest individuals survive to reproduction, passing their DNA to subsequent generations. Thus, in order to assess their capacity in surviving they are assigned a fitness function which will determine their chance of reproducing. See Figure (19.4)

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 19.4: Fitness function

The breeding of new generation is done by using either asexual or sexual reproduction. In asexual reproduction, the DNA is a copy of the parent's DNA, possibly with some random changes called mutation. With sexual reproduction, two of the fitter vectors/strings are chosen and the new vector/string is created by sequentially copying sequences alternately from each parent. Typically only two or three sequences are used, and the point(s) where the copying crosses over to the other parent is chosen at random. This is called crossover. In evolutionary algorithms, the genetic operators considered for reproduction are:

- crossover: we combine the first four characteristics of GW with the last five characteristics of BW.
- mutation: we modify the colour and try a blue rabbit.

They are represented in Figure (19.5).

Figure 19.5: The genetic operators: crossover and mutation

In order to run a GA system for solving a problem, we need to specify a small number of ingredients, often termed preparatory steps. Then, as the DNA within the population changes, the species as a whole changes. That is, the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

species evolves as a result of selective survival of its individuals components.
Evolutionary algorithms contain a population of trial solutions to a problem. Each individual in the population is modelled by a vector / string representing its DNA. It represents a point in the search space which is evaluated by the fitness function to yield a number indicating how good that point is. Thus, the population is evolved by repeatedly selecting the fitter solutions and producing new solution from them. This way, the new solutions are replacing the existing solutions in the population.
The evolutionary steps are listed in the Algorithm (62) and illustrated in Figure (19.6).

Algorithm 62 Evolutionary steps

```
Initialise population
Fitness (evaluation)
repeat
    New population
    Fitness (evaluation)
    Selection
    Crossover
    Mutation
    Accepting
    Test
until stopping criteria
```


This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 19.6: GA running steps

19.1.2 Some optimisation methods

Before detailing differential evolution (DE), we introduce a few alternative optimisation methods (see Witkowski [2011] and Press et al. [1992] for numerical implementation).
For simplicity of exposition we define a few operators used in several optimisation methods. We let $R_U(a,b)$ be a function returning a random uniformly distributed value between a (inclusive) and b (exclusive), and $R_N(\mu,\sigma)$ be a function returning a random normally distributed value drawn from a normal distribution with mean μ and standard deviation σ. Further, we define the mutation operator $M(X,\delta)$, where X is a vector of size N and δ is the mutation strength, as follows:

```
Begin
 i ← R_U(1,N)
 X_i ← ± R_U(0,1) · δ
return X
End
```

19.1.2.1 Random optimisation

Random optimisation is an iterative optimisation method designed for single objective problem which consists in assigning uniformally distributed random values to an individual. In each iteration, the individual is modified by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

adding a normally distributed vector to it, in the case where the resulting individual is better than the source individual. We let $f : \mathbb{R}^N \to \mathbb{R}$ be the fitness function subject to optimisation, and we let $X \in \mathbb{R}^N$ be a position in the search space. The vector X is initialised as follows

$$X \leftarrow [R_{U,1}(0,1), ..., R_{U,N}(0,1)]$$

and the algorithm satisfies

```
Begin
while not  terminationCriterion()  do
    X^' ← X + [R_N,1(μ,σ), ..., R_N,N(μ,σ)]
   if  f(X^') > f(X)  then
       X ← X^'
   end if
end while
End
```

When the value of an individual is subject to a constraint, we consider four strategies where one or more genes do not fit the constraints:

- Dropping the individual altogether: the algorithm leaves the old value of the individual.
- Dropping only the offending gene: the old gene is left.
- Trimming the gene to the constraint: trimming either to the left or to theright value of the constraint.
- Bounce back: if g_{min} and g_{max} are the minimum and maximum values of the gene, respectively, then do:

```
Begin
while not  notConstraints(g)  do
   if  g > g_max  then
       g ← g_max − |g − g_max|
   else
       g ← g_min + |g_min − g|
   end if
end while
End
```

19.1.2.2 Particle swarm optimisation

Particle swarm optimisation (PSO) (see Kennedy et al [1995] [2001]) is an optimisation algorithm simulating swarming/social behaviour of agents (particles) and iteratively trying to improve a candidate solution by a given measure of quality. In this scheme, a single particle is a solution candidate flying through the search space with some velocity, remembering its (local) best position, and learning the best position known to its neighbour particles. While travelling the search space, the particles adjust their speed (both direction and value) based on their personal experiences and on the knowledge of their neighbourhood particles. Since various schemes for determining particle neighbourhood can be imagined, we can discern

- the global neighbourhood: all particles are neighbours to each other.
- the neighbourhood determined by Euclidean distance.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- the neighbourhood determined by normalised Euclidean distance.

where the normalisation process adjust the size of the neighbourhood so that the dimension of the search space is accounted for. This iterative process is expected to move the swarm toward the best solutions.
PSO is a parallel direct search method using NP parameter vector

$$X_i \text{ for } i = 1, .., NP$$

as a population of particles with vector of velocity V. The initialisation is given by

$$X_0, ..., X_{NP} \leftarrow [R_{U,1}(0,1), ..., R_{U,N}(0,1)]$$
$$V_0, ..., V_{NP} \leftarrow [R_{U,1}(0,1), ..., R_{U,N}(0,1)]$$

and the algorithm is as follows:

```
Begin
while not  terminationCriterion()  do
   for  i = 1  to  NP  do
      for  j = 1  to  N  do
          φ_1 ← R_U(0,1)
          φ_2 ← R_U(0,1)
          V_ij^' ← w V_ij + σ ( φ_1 C_1 (best(p_i)_j − X_ij)
          + φ_2 C_2 (best_Nhood(p_i)_j − X_ij) )  (update velocity of the particles)
      end for
      if  V^' > V_max  then
          V^' ← V_max
      end if
      if  V^' < −V_max  then
          V^' ← −V_max
      end if
       X_i^' ← X_i + V_i^'  (move particles to their new positions)
   end for
 X ← X^'
 V ← V^'
end while
End
```

where

- N is the number of dimension of the search space,
- X is a vector of current particle positions with element X_{ij}, $i = 1, .., NP$, $j = 1, ..N$,
- V is a vector of current particle velocities (each velocity is a vector with N elements),
- V_{max} is the maximum possible velocity,
- w is the velocity-weight or inertia factor (how much the particle will base its next velocity on its previous one),
- σ is the position weight determining the importance of the particles position to the next particle velocity,
- ϕ_i, for $i = 1, 2$, are uniform random variables taken as an additional weight when determining the next particle velocity,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- C_1 is the self confidence weight (or self learning rate),
- C_2 is the swarm confidence weight (or neighbourhood learning rate),
- $best_{Nhood}(p)$ is an operator returning the best (most fit) position known to the particle and its neighbourhood (global best position),
- $best(p)$ is an operator returning the most fit position that a particle has visited (current best position).

19.1.2.3 Cross entropy optimisation

The cross entropy optimisation consists of two steps

1. generating a sample population from a distribution.
2. updating the parameters of the random mechanism to produce a better (fitter) sample in the next population.

This behaviour conceptually substitutes the problem of finding an optimal individual to that of iteratively finding a random distribution that generate good individuals. We start by using a random Gaussian distribution to produce variables, and then improve the distribution by means of importance sampling, finding a new distribution from the best samples. We let N be the number of dimensions of the search space, N_{size} be the sample size, $(X_1, .., X_N)$ denotes the optimisation population, and we assume that the population is sorted so that

$$\forall i = 1, .., N-1 \, , \, f(X_i) \geqslant f(X_{i+1})$$

We then define the mean and standard deviation as

$$\begin{aligned} mean(X, N_{sample}) &= \frac{1}{N_{sample}} \sum_{i=0}^{N_{sample}} X_i \\ Std(X, N_{sample}) &= \sqrt{\frac{1}{N_{sample}} \sum_{i=0}^{N_{sample}} (X_i - mean(X, N_{sample}))^2} \end{aligned}$$

Since X is a vector, then $mean(X)$ and $Std(X)$ work on a set of vectors and return a vector of elements, that is, a vector of means and standard deviations of columns of the given input matrix. Additional importance sampling with the size of N_{sample} is performed on the given set. The initialisation is as follow

$$X_0 ... X_{NP} \leftarrow [R_{U,1}(0,1), ..., R_{U,N}(0,1)]$$

and

$$\begin{aligned} \mu &\leftarrow mean(X, N_{sample}) \\ \sigma &\leftarrow Std(X, N_{sample}) \end{aligned}$$

The algorithm is as follow

```
Begin
while not  terminationCriterion()  do
   for  i = 1  to  NP  do
       X_i  ←  [R_N,1(μ_1,σ_1),  ...,  R_N,N(μ_N,σ_N)]
   end for
```


This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

```
 μ  ←  mean(X, N_sample)
 σ  ←  Std(X, N_sample)
end while
End
```

In addition, the values of the mean μ and standard deviation σ can be smoothed over time (from iteration to iteration), improving the convergence of the algorithm.

19.1.2.4 Simulated annealing

Simulated annealing directly addresses the problem of finding global extrema in the presence of large number of undesirable local extrema (see Kirkpatrick [1983]). It is a probabilistic optimisation method inspired by the physical process of annealing metals where a metal is slowly cooled so that its structure is frozen in a minimal energy configuration (see Tsitsiklis et al. [1993]). The essence of the process is skow cooling, allowing ample time for redistribution of the atoms as they lose mobility. In a continuous minimisation algorithm, we want to go greedily for the quick, nearby solution, such as going downhill as far as we can, possibly reaching local minimum. Here, the algorithm is similar to hill climbing where an individual will iteratively go to a better neighbourhood position (picked at random), but additionally an individual may go to a worse position with a probability proportional to its temperature. The probability that an individual will go to a worse position is calculated by the Boltzmann probability factor (distribution) (also called Gibbs distribution)

$$P(E) \sim e^{-\frac{E(pos)}{k_B T}}$$

where $E(pos)$ is the energy at the new position (calculated as a difference of fitness of two positions), k_B is the Botzmann constant $(1.38065052410^{-23} J/K)$, and T is the temperature of the solid. It states that a system in thermal equilibrium at temperature T has its energy probabilistically distributed among all different energy states E. Even at low temperature, there is a chance, very small, for a system to be in a high energy state. Thus, there is a corresponding chance for the system to get out of a local energy minimum in favour of a better, more global one.
The rate at which the solid is frozen is called a cooling schedule. Success or failure is often determined by the choice of the annealing schedule. We consider two variants:

1. $$getTemp(t) = T_{start}(a^t)$$

   where T_{start} is the starting temperature, t is the current iteration, and a is a parameter of the cooling schedule. Further, $a > 0 \wedge a \approx 0$.

2. $$getTemp(t) = T_{start}(1-\epsilon)^{\frac{t}{m}}$$

   where ϵ and m are parameters of the cooling schedule. Further, $0 < \epsilon \leqslant 1$.

The initialisation is as follow:

$$X \leftarrow [R_{U,1}(0,1), .., R_{U,N}(0,1)]$$

and

$$X_{best} \leftarrow X$$

One can apply the ideas of simulated annealing to optimisation problems with continuous N-dimensional control spaces. That is, finding $f(X)$, in the presence of many local minima, where X is an N-dimensional vector. There must be a generator of random changes in the configuration, that is, a procedure for taking a random step from X to $X + \Delta X$. We let $f : \mathbb{R}^N \to \mathbb{R}$, and $M(X, \delta)$ be the mutation operator and define the algorithm as follow:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

```
Begin
while not  terminationCriterion()  do
    X^' ← M(X,δ)
    Δ E ← f(X) − f(X^')
      if  Δ E ⩽ 0  then
          X ← X^'
         if  (f(X)) > f(X_best)  then
             X_best ← X
         end if
      else
          T ← getTemp(t)
         if  R_U(0,1) < e^(Δ E / (k_B t))  then
             X ← X^'
         end if
      end if
 t ← t+1
End
```

19.1.3 Introduction to genetic algorithms

19.1.3.1 Description

Using a digital computer to create artificial intelligence, Nils Barricelli [1954] wrote the first genetic algorithm software (see Simon [2013]). Fraser [1957] worked on the idea of creating a computer model of evolution. Later, Holland [1962] introduced genetic algorithm (GA) as we know it today. That is, is an optimisation technique that produces optimisation of the problem by using natural evolution based on survival of the fittest (see Section (19.1.1.3)) as well as crossover, mutation and recombination. This population based method was extended by Goldberg [1983] [1989]. The methodology is both robust in optimising noisy and non-linear functions, and powerful in automatically providing multiple solutions when they exists.

GA encodes the decision variables of a search problem into finite-length strings of alphabets of certain cardinality. The search space is initialised with a set of solutions called chromosomes, which is a set of genes (the alphabets), and the values of genes are called alleles. For example, in a problem such as the travelling salesman problem (TSP), a chromosome represents a route, and a gene may represent a city (see details in Section (19.1.3.7)).
There exists several ways of translating the sample space of candidate solutions into chromosomes depending on the problem to solve. One approach is to encode each candidate solution in an array of parameter values

$$\text{chromosome} = [p_1, ..., p_N]$$

where p_i is the value of the ith parameter expressed as bit string (sequence of 1's and 0's). The parameters are concatenated end-to-end like genes in a DNA strand to create the chromosomes. The latter can also include permutations, real numbers and many other objects. Quality and fitness of a chromosome is measured by a fitness function. The behaviour of genetic algorithm is determined by exploration and exploitation with the help of operators like reproduction (selection), crossover (recombination) and mutation. GAs rely on a population of candidate solutions.

Once the problem is encoded in a chromosomal manner and a fitness measure for discriminating good solutions from bad ones has been chosen, we can evolve solutions to the search problem. The general procedure for genetic algorithm is (see Witten et al. [2005]):

1. START (initialisation): Generate random population across the search space. Note, domain-specific knowledge or other information can be easily incorporated.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. FITNESS (evaluation): Evaluate the fitness $f(x)$ of each chromosome x in the population.
3. NEW POPULATION: Create a new population by repeating following steps until the new population is complete
   - SELECTION: It allocates more copies of those solutions with higher fitness values. The main idea of selection is to prefer better solutions to worse ones. Some examples are roulette-wheel selection, stochastic universal selection, ranking selection and tournament selection.
   - REPRODUCTION (or recombination): Parents chromosomes are selected from population according to their fitness to crossover and produce new offspring.
   - CROSSOVER: Crossover operator produces new two offspring from selected two parents based crossover probability. See Figure (19.7).
   - MUTATION: Mutation operator produces new offspring by mutating single bit position in chromosome. It usually involves one or more changes being made to an individual's trait or traits. That is, mutation performs a random walk in the vicinity of a candidate solution. It is used to maintain genetic diversity. See Figure (19.7).
4. ACCEPTING: Place new offspring in the new population.
5. REPLACE: Use new generated population for a further run of the algorithm. Many replacement techniques exist such as elitist replacement, generation-wise replacement and steady-state replacement methods.
6. TEST: If the end condition is satisfied, stop, and return the best solution in current population.
7. LOOP: Go to stop.

Figure 19.7: GA mutation and crossover

The fitness function determines how the chromosomes will change over time. As such, it needs to accurately score the chromosomes based on a range of fitness values so that a complete solution can be distinguished from partial solutions.

Performance of GA depends on various parameters like population size, crossover and mutation rate and computation time. Performance of genetic algorithm can be optimised by parallel genetic algorithm. Parallel Genetic Algorithm (PGA) is an algorithm that works by dividing a large problem into smaller tasks. Genetic algorithm with parallel processing reduces the computation time.

19.1.3.2 Genetic operators

A genetic operator is an operator used in genetic algorithms to guide the algorithm towards a solution to a given problem. The common genetic operators in GA are selection, recombination, and mutation. We briefly describe them.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

19.1.3.2.1 Selection Selection procedures can be broadly classified into two classes as follows:

1. Fitness proportionate selection: It includes methods such as roulette-wheel selection (see Holland [1975]) and stochastic universal selection (see Baker [1985]). In roulette-wheel selection, each individual in the population is assigned a roulette wheel slot sized in proportion to its fitness. That is, in the biased roulette wheel, good solutions have a larger slot size than the less fit solutions. The roulette wheel is spun to obtain a reproduction candidate. The roulette-wheel selection scheme can be implemented as follows:

   (a) Evaluate the fitness f_i of each individual in the population.
   (b) Compute the probability p_i of selecting each member of the population: $p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$ where n is the population size.
   (c) Calculate the cumulative probability q_i for each individual: $q_i = \sum_{j=1}^{i} p_j$.
   (d) Generate a uniform random number $r \in (0, 1]$.
   (e) If $r < q_1$ then select the first chromosome x_1 else select the individual x_i such that $q_{i-1} < r \leqslant q_i$.
   (f) Repeat steps $4 - 5$ n times to create n candidates in the mating pool.

2. Ordinal Selection: It includes methods such as tournament selection (Goldberg [1989]), and truncation selection (see Muhlenbein et al. [1993]). In tournament selection, s chromosomes are chosen at random (either with or without replacement) and entered into a tournament against each other. The fittest individual in the group of k chromosomes wins the tournament and is selected as the parent. The most widely used value of s is 2. Using this selection scheme, n tournaments are required to choose n individuals. In truncation selection, the top $(\frac{1}{s})$th of the individuals get s copies each in the mating pool.

19.1.3.2.2 Recombination After selection, individuals from the mating pool are recombined (or crossed over) to create new, hopefully better, offspring. In the GA literature, many crossover methods have been designed (see Goldberg [1989]). Many of the recombination operators used in the literature are problem-specific and in this section we will introduce a few generic (problem independent) crossover operators. It should be noted that while for hard search problems, many of the following operators are not scalable, they are very useful as a first option. Significant success have been obtained in designing scalable recombination operators that adapt linkage.
In most recombination operators, two individuals are randomly selected and are recombined with a probability p_c called the crossover probability. That is, a uniform random number r is generated and if $r \leqslant p_c$ the two randomly selected individuals undergo recombination. Otherwise, if $r > p_c$ the two offspring are simply copies of their parents. The value of p_c can either be set experimentally, or can be set based on schema-theorem principles.
Some of the methods for crossover are:

- k-point Crossover: One-point, and two-point crossovers are the simplest and most widely applied crossover methods. In one-point crossover (see Figure (19.8), a crossover site is selected at random over the string length, and the alleles on one side of the site are exchanged between the individuals. In two-point crossover, two crossover sites are randomly selected. The alleles between the two sites are exchanged between the two randomly paired individuals. The concept of one-point crossover can be extended to k-point crossover, where k crossover points are used, rather than just one or two.

- Uniform Crossover: Another common recombination operator is uniform crossover. In uniform crossover, every allele is exchanged between the a pair of randomly selected chromosomes with a certain probability p_e known as the swapping probability. Usually the swapping probability value is taken to be 0.5.

- Uniform order-Based Crossover: The k-point and uniform crossover methods described above are not well suited for search problems with permutation codes such as the ones used in the travelling salesman problem (TSP). They often create offspring that represent invalid solutions for the search problem. Therefore, when solving search problems with permutation codes, a problem-specific repair mechanism is often required (and used) in conjunction with the above recombination methods to always create valid candidate solutions.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 19.8: GA points crossover

19.1.3.2.3 Mutation If we use a crossover operator, such as one-point crossover, we may get better and better chromosomes. However, if the two parents (or worse, the entire population) have the same allele at a given gene, then one-point crossover will not change that. That is, the gene will have the same allele forever. Mutation is designed to overcome this problem in order to add diversity to the population and ensure that it is possible to explore the entire search space. In evolutionary strategies, mutation is the primary variation/search operator. Unlike evolutionary strategies, mutation is often the secondary operator in GAs, performed with a low probability. One of the most common mutations is the bit-flip mutation. In bitwise mutation, each bit in a binary string is changed (a 0 is converted to 1, and vice versa) with a certain probability, p_m, known as the mutation probability. As mentioned earlier, mutation performs a random walk in the vicinity of the individual. Other mutation operators, such as problem-specific ones, can also be developed and are often used in the literature.

19.1.3.2.4 Replacement Once the new offspring solutions are created using crossover and mutation, we need to introduce them into the parental population. Recall, the parent chromosomes have already been selected according to their fitness, so we are hoping that the children (which includes parents which did not undergo crossover) are among the fittest in the population and so we would hope that the population will gradually, on average, increase its fitness. Some of the most common replacement techniques are as follows:

- Delete-all: It deletes all the members of the current population and replaces them with the same number of chromosomes that have just been created. This is the most common technique and is the technique of choice for most people due to its relative ease of implementation. It is also parameter-free, which is not the case for some other methods.
- Steady-state: It deletes n old members and replaces them with n new members. The number to delete and replace, n, at any one time is a parameter to this deletion technique. We also need to decide which members to delete from the current population. Should we delete the worst individuals, pick them at random or delete the chromosomes that we used as parents? Again, this is a parameter to this technique.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Steady-state-no-duplicates: This is the same as the steady-state technique but the algorithm checks that no duplicate chromosomes are added to the population. This adds to the computational overhead but can mean that more of the search space is explored.

19.1.3.3 Schema theories

Schema theories are among the oldest and the best known models of evolutionary algorithms (see Holland [1973] [1975]). They are based on the idea of partitioning the search space into subsets, called schemata. A schema is a template that identifies a subset of strings with similarities at certain string positions. Schemata are a special case of cylinder sets, and hence form a topological space. They are concerned with modelling and explaining the dynamics of the distribution of the population over the schemata.
The schema theorem (see Holland [1973]) requires the vector representation and fitness function be designed so that the required solution can be composed of short fragments of vectors which, if present in a vector, give it a relatively high fitness regardless of the contents of the rest of the vector. That is, short and low-order schemata with above-average fitness increase exponentially in frequency in successive generations. These are known as building blocks. They can be thought of as collections of genes which work well together. Assuming that building blocks exist, GA can progressively select the vectors with building blocks and using the crossover operator gradually splice these together until the population contains vectors which are substantially correct.
Nonetheless, this theorem was widely taken to be the foundation for explanations of the power of genetic algorithms. However, while the schema theorem holds under the assumption of a genetic algorithm that maintains an infinitely large population, it does not always carry over to (finite) practice. That is, due to sampling error in the initial population, genetic algorithms may converge on schemata that have no selective advantage.
Note, Stephens et al. [1999] provided exact information about the distribution of the population at the next generation in terms of quantities measured at the current generation, without having to actually run the algorithm.

19.1.3.4 Dual representation and interpretation function

The structural recombination happening in the crossover operator relies on two distinct representational spaces. See Figure (19.9). These spaces are:

1. Recombination space: It is defined over a set of fixed-length binary strings. It is the set of structures to which the genetic operators are applied. The actual search occurs in that space.

2. Evaluation space: It involves a problem-dependent representation. It is the set of structures whose ability to perform a task is evaluated.

The interpretation function maps between these two representational spaces. We will in Section (19.3) that we might be interested in evolving trees, graphs, and even networks (see Section (20.1)). Since any set of finite-length bit strings cannot represent all possible networks, the evaluation space is restricted to a predetermined set of networks. However, by design, the dual representation scheme allows crossover operations by implicitly assuming that the bit strings created by the dynamics of the GA will map to successively better networks.
This is an important feature when it is unclear how to search the evaluation space directly and when the interpretation function is efficient. However, the choice of the interpretation function may exclude several interesting and useful networks. Thus, having a dual representation makes it difficult for crossover to be an appropriate evolutionary operator for some tasks. Goldberg [1989] argued that crossover tend to recombine short, connected substrings of the bit string representation corresponding to above-average task solutions. These substrings are called building blocks, making explicit the intuition that larger structures with high fitness are built out of smaller structures with moderate fitness. Crossover tends to be most effective in environments where the fitness of a member of the population is reasonably correlated with the expected ability of its representational components (see Goldberg [1989c]). However, environments where this is not true are called deceptive (see Goldberg [1989b]).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 19.9: GA dual representation scheme

19.1.3.5 Fitness landscapes

We saw in Section (19.1.3.1) that individuals in a population were represented as fixed length bitstrings. As such, the total number of possible individuals is equal to the total number permutations of the bitstring, or 2^n where n is the of bits in the bitstring. We can plot these individuals as points on a horizontal plane such that the distant between individuals is equal to the number of bit changes required to get from the one individual to the other (the hamming distance). We can then give these points a vertical component proportional to the fitness values of the individuals that they represent. We end up with a fitness landscape (FL), where the peaks represent location of high fitness, and the valleys locations of poor fitness. The actual ruggedness of this landscape depends on the difficulty of the fitness function. A particularly difficult function might result in a fitness landscape which is flat and virtually featureless, with the exception of a few widely distributed spires (see Figure (19.10)). Conversely a very easy fitness function might consist of a mountain with a single peak (see Figure (19.10)). When we generate an initial random population of individuals, we are effectively placing a mesh over the fitness landscape, simultaneously sampling the fitness function at different places. Thus, genetic operators such as mutation and crossover provide a means for exploring this landscape. Single bit mutation allows us to move one point at a time across the landscape. Crossover and multi-bit mutation allow us to take giant leaps across the landscape. It is implicitly assumed that if we select individuals of higher fitness, then by applying the genetic operators, we should end up on the landscape at a point of high fitness. That is, there should be some correlation between parents and children. This notion is called auto-correlation.

19.1.3.6 Multimodal functions

Many practical search and optimisation problems require the investigation of multiple local optima (or multiple objective optimisation). However, genetic algorithms (GAs) utilise populations of individual hypotheses that converge over time to a single optimum, even within a multimodal domain. They converge over time to one hypothesis within the domain. Thus, they are unable to maintain rules of secondary importance due to population convergence to the best value.
One solution is to weight each function f_i with the weighting factor w_i so that

$$cost = \sum_{i=1}^{N} w_i f_i \text{ with } \sum_{i=1}^{N} w_i = 1$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 19.10: GA difficult and easy fitness landscape

but it is very difficult to estimate the appropriate weights w_i, $i = 1, ..., N$. Acknowledging that there is no single optimal solution, but a set of optimal solutions called the Pareto optimal solutions, we can find as many solutions as possible. The idea is to order the solutions based on some dominance criteria. A solution is Pareto optimal if no other solution dominates that solution with respect to the fitness functions. This can be achieved by running a GA for several combinations of the weighted cost function. Schaffer [1984] implemented such a multi-objective GA called vector-evaluated GA (VEGA), but it tended to converge to a single solution. To avoid such convergence to a single solution, some population diversity mechanisms were proposed. They enable genetic algorithms to identify multiple optima within multimodal domains by maintaining population members within the niches defined by the multiple optima (see Miller et al. [1996]).
In nature, there are different sub-spaces (niches) supporting different types of life (species or organisms). The right number of organisms is required to sustain life within a given niche. In a multimodal domain, each peak can be thought of as a niche supporting a certain number of concepts. The number of individuals supported by a niche is directly proportional to its carrying capacity measured by the niche peak's fitness relative to other niche peaks' fitness in the domain. In the niche proportionate population, the niche are populated according to their fitness relative to the other peaks. However, the niche peak locations of a multimodal domain are not known a priori. That is, the niche location is uncertain. Thus, the GAs must both identify the niche locations and proportionally populate the niches according to their fitness.
Some examples of population diversity mechanisms are:

- Crowding
- Sharing
- Dynamic niching

19.1.3.6.1 Sharing Sharing was introduced by Holland [1975] and extended by Goldberg et al. [1987]. They developed a method of sharing functions to permit the formation of stable sub-populations of different strings within a genetic algorithm, thereby permitting the parallel investigation of many peaks. The idea being to reduce the fitness of individuals having highly similar members within the population by rewarding individuals that uniquely exploit areas of the domain and discouraging the redundant ones. By causing population diversity pressure, it helps maintain population members at local optima.
The shared fitness of the ith individual is given by

$$f_{sh,i} = \frac{f_i}{m_i}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where f_i is the raw fitness of the individual, and m_i is the niche count defining the amount of overlap (sharing) of the ith individual with the rest of the population. The latter is calculated by summing a sharing function over all members of the population:

$$m_i = \sum_{j=1}^{N} sh(d_{i,j})$$

where $d_{i,j}$ is the distance between the ith individual and the jth one in the population, determined by a similarity metric

$$sh(x) = \begin{cases} 1 - (\frac{x}{\sigma_{sh}})^{\alpha_{sh}} \text{ if } x < \sigma_{sh} \\ 0 \text{ otherwise} \end{cases}$$

In general, the parameter α_{sh} is set to 1, resulting in the triangular sharing function. It states that if the sharing function determines that the distance is within a fixed radius σ_{sh}, it returns a value between 0 and 1 that increases with greater similarity (decreasing $d_{i,j}$). Otherwise it returns 0. Deb et al. [1989] devised formulas for estimating the appropriate value for σ_{sh} given the expected number of peaks in the multimodal domain q and the hyper-volume of the entire domain space.
The similarity metric can be used on

- phenotype: it uses domain knowledge (if it is sufficient) to calculate a more meaningful distance measurement.
- genotype similarity: it is a domain independent metric such as the Hamming distance between the genotype bit-string representation.

In both cases, the metric must return a distance $d_{i,j}$ that decreases as similarity increases. Some of the problems with this approach are as follows:

- it may be difficult to accurately estimate the number of expected peaks q in the domain.
- estimating σ_{sh} can be complicated by irregular peaks.
- calculating the shared fitness of the individuals incurs an additional complexity resulting from comparing each individual against other members of the population to estimate the niche count m.

19.1.3.7 Solving the TSP

We have presented discrete optimisation in Section (18.1.3) and introduced the travelling salesman problem (TSP). We are now going to discuss its solution when using genetic algorithm. We follow the approach presented by Haupt et al. [2004] and modified by Carr [2014].
We assume that the salesman visit N cities via the shortest possible route and that each city is visited exactly once. We let the problem be closed, that is, the salesman ends its journey where he started. We denote the cities with the set $\{c_i; i = 1, ..., N\}$ where $i = 1$ is the starting point. Thus, the chromosome for the genetic algorithm is the different permutations of the integer 1 through N. We assume the symmetric problem where the distance d from city c_i to city c_j is

$$d(c_i, c_j) = d(c_j, c_i) \text{ for all } i \in [1, N] \text{ and } j \in [1, N]$$

Since each city must be represented once and only once, we need to modify the crossover and mutation operators. There are several ways for modifying these operators to accommodate permutations (see Goldberg [1989]): the partially matched crossover (PMX), the order crossover (OX) (it attempts to maintain the order of integers as if the chromosome vector was wrapped in a circle, so that the last element is followed by the first one), the cycle crossover (CX). We briefly discuss the CX method. The method described by Haupt et al. [2004]) starts at the left of the string and exchange the first two digits, which leads to a duplicate integer. Once identified, the latter is exchanged with the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

other offspring. This process is repeated until there is no repeated integers in either string. Carr [2014] proposed a slight variation of the method. For simplicity of exposition we consider chromosomes of length 6. Given two parent chromosomes, we choose a random location in the length of the chromosome to create the offspring. The random location is marked with bars in Table (19.1).

Parents	Step1	Step2	Step3	Step4
415|3|26	4152|2|6	4|1|5216	|4|45216	345216
346|2|15	3463|1|5	3|4|6325	|3|16325	416325

Table 19.1: Example of cycle of crossover

Unless the swapped integers are the same value, each offspring has a duplicate integer. In step1 we switch the duplicate integer in the first offspring (3) with the integer in the same location in the second offspring (2). There is now a duplicate of a different integer. We repeat the process until there are no duplicate in the first or second offspring. In step2 we switch the duplicate integer in the first offspring (2) with the integer in the same location in the second offspring (1). In step3 we switch the duplicate integer in the first offspring (1) with the integer in the same location in the second offspring (4). In step4 we switch the duplicate integer in the first offspring (4) with the integer in the same location in the second offspring (3). We stop as we have reached our goal.
The modified mutation operator randomly chooses two integers in a chromosome from the new generation and swap them.
We suppose that the N cities are listed in the order $c_1 \to c_2 \to \cdots \to c_N$ in a given chromosome. We represent them by randomly placing N points on the xy-plane in the 1×1 square between $(0,0)$ and $(1,1)$. Since we want to minimise the total distance travelled by the salesman, we let the fitness function be that total distance

$$D = \sum_{k=1}^{N} d(c_k, c_{k+1})$$

where $N+1 = 1$ since the problem is closed. We let the pair (x_i, y_i) denotes the coordinates of the ith city. Then, the fitness function becomes

$$D = \sum_{k=1}^{N} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2}$$

Carr [2014] improved on Haupt's code by keeping the fitter half of the current population and generating the other half of the new generation through selection and crossover. The mutation operator acts on each member of the new generation with probability 0.05. The number of cities is $N = 20$. The calculation of the probability distribution for the selection operator is given in Algorithm (63).

Algorithm 63 Probability distribution for the selection operator

Require: $Npop$ is the number of population, $Ncity$ is the number of city, $keep$ is the population members that survive

```
odds = 1 initialise odds
for n = 2 to keep do
    odds = [odds n × ones(1, n)]
end for
Nodds = length(odds)
odds = keep − odds + 1 odds determine the probabilities for being parents
```


This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The current generation of chromosomes is sorted from the highest fitness to lowest fitness from 1 to 20 and only the fitter half (10 chromosomes) is kept. Note, the last line reverse the probability distribution of the pool of chromosomes that are chosen for reproduction. Odds is a vector containing the integers from 1 to 10, with ten instances of integer 1, nine instances of integer 2, down to one instance of integer 10. We then select randomly a parent from the vector odds, so that the probability of the fittest chromosome is $\frac{10}{55}$ and the probability of the least fit chromosome is $\frac{1}{55}$. If we remove the last line of code the least fit chromosome has the highest probability of becoming a parent and the fittest one has the least probability. Results are obtained for $10,000$ iterations. Since the optimal solution should have no crossing paths, we can plot the solution and check.

19.1.3.8 Advantages and disadvantages

Some authors have combined genetic algorithms with other optimisation algorithms as well as other branches of evolutionary computation, such as neural networks. For instance, GA can be combined with the calculus-based hill-climbing methods (see Section (18.3)). Each of the fittest chromosomes found is then used as a starting point for the hill-climbing algorithm, which follows the gradient of the fitness function from there to a local optimum.
Parallel computing has made genetic algorithms even more appealing because it can alleviate their main drawback such as the lack of speed. It allows for more iterations and more total runs, which increases both the chances of finding better solutions and the certainty that the solutions found are the optimal ones.
Some of the advantages and disadvantages of GA are:

1. Advantages of GA:
   - GA is faster and more efficient as compared to the traditional methods.
   - It has very good parallel capabilities.
   - It optimises both continuous and discrete functions and also multi-objective problem.
   - It is useful when the search space is very large and there are a large number of parameters involved.
2. Disadvantages of GA:
   - Fitness values are calculated repeatedly which might be computationally expensive for some problems.
   - Being stochastic, there is no guarantee of the optimality or the quality of the solution.

When using GAs with classification algorithms, some of the issues and challenges are

- Local convergence: GA sometimes converge on local optima, incorrect peak populate due to sampling errors. One needs to modify the selection pressure to prevent premature convergence.
- Parameter setting: GA parameters such as selection rate, chromosome length, population size, crossover probability, mutation probability and total number of generations has a large impact on performance of genetic algorithm. Setting these parameters is a complex task.

The power of GAs is being demonstrated for an increasing range of applications: financial, imaging, VLSI circuit layout, gas pipeline control and production scheduling. One of the most intriguing uses of GAs is automatic program generation (see Section (19.3)).

19.1.4 Introduction to differential evolution

19.1.4.1 The DE algorithm

According to Feoktistov [2006], Differential Evolution (DE) is first and foremost an optimisation algorithm, and in particular, one of the most powerful tools for global optimisation, regardless of it simplicity. It is so named because it

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

identifies differences in individuals through the use of a simple and fast linear operator (differentiation), and in doing so, realises the evolution of a population of individuals in some intelligent manner. That is, the main characteristic of DE is an adaptive scaling of step sizes resulting in fast convergence behaviour. The Differential Evolution (DE) proposed by Storn and Price [1995] is an algorithm that can find approximate solutions to nonlinear programming problems. It is a parallel direct search method using NP parameter vectors, where each vector (or individual) is of dimension D equals the number of objective function parameters. The vectors

$$X_{i,G} \text{ for } i = 0, .., NP - 1$$

forms a population for each generation G, like any evolutionary algorithms. The initial vector population is chosen randomly, covering the entire parameter space. The number of vectors NP is a function of the dimension D, such as $NP \in [5D, 10D]$ but NP must be at least 4 to ensure that DE will have enough mutually different vectors to play with (see Storn et al. [1997]). Each of the NP parameter vectors undergoes mutation, recombination and selection.

19.1.4.1.1 The mutation The role of mutation is to explore the parameter space by expanding the search space giving its name to the DE algorithm. For a given parameter vector $X_{i,G}$ called the Target vector, the DE generates a Donor vector V made of three or more independent parent vectors $X_{r_l,G}$ for $l = 1, 2, ..$ where r_l is an integer chosen randomly from the interval $[0, NP - 1]$ and different from the running index i. In the spirit of Wright [1991], the main idea is to perturbate a Base vector $\hat{V}$ with a weighted difference vector (called differential vectors)

$$V = \hat{V} + F \sum_{l=1} \left(X_{r_{2l-1},G} - X_{r_{2l},G} \right) \tag{19.1.1}$$

where the mutation factor F is a constant taking values in $[0, 2]$ and scaling the influence of the set of pairs of solutions selected to calculate the mutation value. Most of the time, the Base vector is defined as the arithmetical crossover operator

$$\hat{V} = \lambda X_{best,G} + (1 - \lambda) X_{r_1,G}$$

where $\lambda \in [0, 1]$ allows for a linear combination between the best element $X_{best,G}$ of the parent population vectors and a randomly selected vector $X_{r_1,G}$. It is called a global selection when $\lambda = 1$ while when $\lambda = 0$ the base vector is the same as the target vector, $X_{r_1,G} = X_{i,G}$ and we get a local selection. In the special case where the mutation factor is set to zero, the mutation operator becomes a crossover operator. Figure (19.11) displays graphically the solution space and how the mutation vector V is realised. Note, the parameter vector X_{r_3} that currently resides outside of the solution space demonstrates how the mutation procedure allows for exploration of alternative solution spaces and therefore, how the algorithm is able to reliably converge to global minima.

19.1.4.1.2 The recombination Recombination incorporates successful solutions from the previous generation. That is, according to a rule, we combine elements of the Target vector $X_{i,G}$ with elements of the Donor vector $V_{i,G}$ to create an offspring called the Trial vector $U_{i,G}$. In order to increase the diversity of the parameter vectors, elements of the Donor vector enter the Trial vector with probability CR. In the DE algorithm, each element of the Trial vector satisfies

$$U_{i,G}(j) = \begin{cases} V_{i,G}(j) \text{ for } j = n_r \mod dim, (n_r + 1) \mod dim, ..., (n_r + L - 1) \mod dim \\ X_{i,G}(j) \text{ for all other } j \in [0, .., NP - 1] \end{cases}$$

where $< n_r >_{dim} = n_r \mod dim$ is the modulo of n_r with modulus dim, dim is the dimension of the vector V (here $dim = N$), and the starting index n_r is a randomly chosen integer from the interval $[0, dim - 1]$. Hence, a certain sequence of the element of U is equal to the element of V, while the other elements get the original element of $X_{i,G}$. We only choose a subgroup of parameters for recombination, enhancing the search in parameter space. The integer L denotes the number of parameters that are going to be exchanged and is drawn from the interval $[1, dim]$ with probability

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 19.11: Creation of differential mutation vector. (Feoktistov [2006])

$$P(L > \nu) = (CR)^{\nu}\,,\, \nu > 0$$

The random decisions for both n_r and L are made anew at each new generation G. The term $CR \in [0,1]$ is the crossover factor controlling the influence of the parent in the generation of the offspring. A higher value means less influence from the parent. Generally, values of CR in the range $[0.8,1]$ lead to good results (see Lampinen et al. [2004]). Most of the time, the mutation operator in Section (19.1.4.1.1) is sufficient and one can directly set the Trial vector equal to the Donor vector.

19.1.4.1.3 The selection Unlike the previous two procedures, the selection procedure is typically deterministic under Differential Evolution. The tournament selection only needs part of the whole population to calculate an individual selection probability where subgroups may contain two or more individuals. In the DE algorithm, the selection is deterministic between the parent and the child. The best of them remain in the next population. We compute the objective function with the original vector $X_{i,G}$ and the newly created vector $U_{i,G}$. If the value of the latter is smaller than that of the former, the new Target vector $X_{i,G+1}$ is set to $U_{i,G}$ otherwise $X_{i,G}$ is retained

$$X_{i,G+1} = \begin{cases} U_{i,G} \text{ if } H(U_{i,G}) \leqslant H(X_{i,G})\,,\, i = 0,..,NP-1 \\ X_{i,G} \text{ otherwise} \end{cases}$$

This method, as applied in many Evolutionary Algorithms, ensures that the population fitness will always increase or remain constant through each generation. Mutation, recombination and selection continue until some stopping criterion is reached. The mutation-selection cycle is similar to the prediction-correction step in the EM algorithm or in the filtering problems. Figure (19.12) gives a graphical account of the tournament selection process.

19.1.4.1.4 Simple convergence criteria We allow for different convergence criterion in such a way that if one of them is reached, the algorithm terminates. We let $f^* = f_{min}(G)$ be the fittest design in the population so far, and we define

$$f_{a,G} = \frac{1}{NP}\sum_{i=0}^{NP-1} f(X_{i,G})$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 19.12: Tournament selection. (Feoktistov, [[2006]])

as the average objective value at generation G. The variance of the objective value at generation G is given by

$$f_{v,G} = \frac{1}{NP}\sum_{i=0}^{NP-1}\left(f(X_{i,G}) - f_{a,G}\right)^2 = \frac{1}{NP}\sum_{i=0}^{NP-1} f^2(X_{i,G}) - f_{a,G}^2$$

Then, when the percentage difference between the average value and the best design reaches a specified small value ϵ_1

$$\frac{|f_{a,G} - f_{min}(G)|}{|f_{a,G}|} \times 100 \leqslant \epsilon_1$$

we terminate the algorithm. Also, we let $f_{min}(G-1)$ be the fittest design in the previous generation $(G-1)$, and consider as a criterion the difference

$$|f_{min}(G-1) - f_{min}(G)| < \epsilon_2$$

where ϵ_2 is user defined. In that case, the DE algorithm will continue until there is no appreciable improvement in the minimum fitness value or some predefined maximum number of iterations is reached.

19.1.4.2 Pseudocode

We now present the pseudo code of a standard DE algorithm.

```
Initialise vectors of the population NP
Evaluate the cost of each vector
    for i=0 to Gmax do
        repeat
            Select some distinct vectors randomly
            Perform mutation
            Perform recombination
            Perform selection
                if offspring is better than main parent then
                replace main parent in the population
                end if
        until population is completed
        Apply convergence criterions
    next i
```


This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

19.1.4.3 The strategies

Over the years, Storn and Price as well as a large number of other authors made improvement to the DE model so that there are now many different DE models (see Price et al. [2005], Storn [2008]). These models vary in the type of recombination operator used as well as in the number and type of solutions used to calculate the mutation values. We are now going to list the main schemes perturbating the base vector for mutation.

19.1.4.3.1 Scheme DE1 To improve convergence on a set of optimisation problems using the Scheme DE4 introduced in Section (19.1.4.3.4), Storn and Price considered for each vector $X_{i,G}$ a trial vector V generated according to the rule

$$V = X_{r_1,G} + F\big(X_{r_2,G} - X_{r_3,G}\big)$$

where the integer r_l for $l = 1, 2, 3$ are chosen randomly in the interval $[0, NP - 1]$ and are different from the running index i. Also, F is a scalar controlling the amplification of the differential variation $X_{r_2,G} - X_{r_3,G}$. We can slightly modify the scheme by writing

$$V = X_{r_1,G} + F\big(X_{r_2,G} + X_{r_3,G} - X_{r_4,G} - X_{r_5,G}\big)$$

More generally, the base vector can be expressed as a linear combination of other distinct vectors, as in Equation (19.1.1). As a simple rule, the differential weight F is usually in the range $[0.5, 1]$ (see Lampinen et al. [2004]). The population size should be between $3N$ and $10N$ and generally we increase NP if misconvergence happens. In the case where NP is increased we should decrease F.

19.1.4.3.2 Scheme DE2 Similarly to the Scheme DE1 in Section (19.1.4.3.1), the trial vector V is generated according to the rule

$$V = X_{i,G} + \lambda\big(X_{best,G} - X_{i,G}\big) + F\big(X_{r_2,G} - X_{r_3,G}\big)$$

where λ is a scalar controlling the amplification of the differential variation $X_{best,G} - X_{i,G}$. It enhances the greediness of the scheme by introducing the current best vector $X_{best,G}$. It is useful for non-critical objective functions, that is, when the global minimum is relatively easy to find. It gives a balance between robustness and fast convergence.

19.1.4.3.3 Scheme DE3 In the same spirit, Mezura-Montes et al. [2006] also modified the Scheme DE1 in Section (19.1.4.3.1) by incorporating information of the best solution as well as information of the current parent in the current population to define the new search direction.

$$V = X_{r_3,G} + \lambda\big(X_{best,G} - X_{r_2,G}\big) + F\big(X_{i,G} - X_{r_1,G}\big)$$

where λ is a scalar controlling the amplification of the differential variation $X_{best,G} - X_{r_2,G}$. This scheme has the same properties as the Scheme DE2 in Section (19.1.4.3.2).

19.1.4.3.4 Scheme DE4 The oldest strategy developed by Storn and Price [1997] is for the trial vector V to be generated according to the rule

$$V = X_{best,G} + F\big(X_{r_1,G} - X_{r_2,G}\big)$$

where the weight F is a scalar. However, in that setting they found several optimisation problems where misconvergence occurred. To improve the convergence, Price et al. [2005] proposed to perturb the differential weight F by introducing the Dither and Jitter schemes. In the former, Karaboga et al. [2004] randomised the weight as follow

$$\lambda_G = F_l + U_G(0,1)\big(F_u - F_l\big)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where F_l is a lower weight, F_u is an upper weight, and $U_G(0,1)$ is uniformly distributed in the range $[0,1]$ for each generation G. In the latter, the random weight is picked for each element $j = 0, ..., D-1$ of the vector being updated

$$\lambda_j = F \times \left[1 + \delta \times (U_j(0,1) - \frac{1}{2})\right]$$

where δ determines the scale of perturbation. It must be small, and is usually set to $\delta = 0.0001$. For example, we can consider the strategy

$$V = X_{best,G} + \lambda_j \big(X_{r_1,G} - X_{r_2,G}\big)$$

It is a jitter which add fluctuation to the random target. The jitter is the time variation of a periodic signal in electronics and telecommunications (swing dancer). It is tailored for small population sizes and fast convergence. We can also modify the scheme by doing

$$V = X_{best,G} + \lambda_j \big(X_{r_1,G} + X_{r_2,G} - X_{r_3,G} - X_{r_4,G}\big)$$

Going one step further, Storn [2000] combined the jitter scheme with the dither one, getting

$$\lambda_{j,G} = \Big(F_l + U_G(0,1)\big(F_u - F_l\big)\Big)\left[1 + \delta \times (U_j(0,1) - \frac{1}{2})\right]$$

19.1.4.3.5 Scheme DE5 Das et al. [2005] improved the DE's convergence by applying the dither scheme to every difference vector. Similarly to the Scheme DE1 in Section (19.1.4.3.1), the trial vector V is generated according to the rule

$$V = X_{r_1,G} + \lambda_{d,i} \big(X_{r_2,G} - X_{r_3,G}\big)$$

where $\lambda_{d,i}$ is a computer dithering factor

$$\lambda_{d,i} = F + U_i(0,1) \times (1 - F)\,, i = 0, ..., NP-1$$

also written

$$\lambda_{d,i} = F_l + U_i(0,1)\big(F_u - F_l\big)\,, i = 0, ..., NP-1$$

where $U_i(0,1)$ is uniformly distributed in the range $[0,1]$. It is a per-vector dither, making the Scheme DE1 more robust. Note, as discussed in Section (19.1.4.3.4) we can also have

$$\lambda_{d,G} = F + U_G(0,1) \times (1 - F)$$

which is a per-generation dither factor. In that algorithm, choosing $F = 0.3$ is a good start. As explained by Pedersen [2010], the dither and jitter schemes are similar, except that the dither draws a random weight once for each agent-vector to be updated and the jitter draws a random weight for each element of that vector. To see this, we simply set $F_l = F \times (1 - \frac{\delta}{2})$ and $F_u = F \times (1 + \frac{\delta}{2})$. The dither can simply be rewritten as

$$\lambda_{d,i} \sim U_i(F_l, F_u)\,, i = 0, ..., NP-1$$

where $U_i(F_l, F_u)$ is uniformly distributed in the range $[F_l, F_u]$, and the jitter can simply be rewritten as

$$\lambda_j \sim U_j(F \times (1 - \frac{\delta}{2}), F \times (1 + \frac{\delta}{2}))\,, j = 0, ..., D-1$$

where $U_i(F \times (1 - \frac{\delta}{2}), F \times (1 + \frac{\delta}{2}))$ is uniformly distributed in the range $[F \times (1 - \frac{\delta}{2}), F \times (1 + \frac{\delta}{2})]$. Defining a midpoint F_{mid} and a range F_{range}, the dither becomes

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\lambda_{d,i} \sim U_i(F_{mid} - F_{range}, F_{mid} + F_{range}) \, , i = 0, ..., NP - 1$$

and the jitter becomes

$$\lambda_j \sim U_j(F_{mid} - F_{range}, F_{mid} + F_{range}) \, , j = 0, ..., D - 1$$

We therefore need to select two parameters, F_{mid} and F_{range}, to determine the limits of perturbation for the weight F. Pedersen chose the midpoint in the interval $F_{mid} \in [0, 2]$ and the range in the interval $F_{range} \in [0, 3]$ allowing for negative differential weights to occur. Thus, perturbing behavioural parameters introduce new parameters in the form of the boundaries for the stochastic sampling ranges. Further, when the behavioural parameters are completely random, it takes statistically a lot of samples before finding an optimum.

19.1.4.3.6 Scheme DE6 A more complex algorithm than the Scheme DE1 in Section (19.1.4.3.1) is to introduce the choice between two strategies. In the either-or algorithm, the trial vector V is generated according to the rule

$$V = \begin{cases} \Big(X_{r_1,G} + F\big(X_{r_2,G} - X_{r_3,G}\big)\Big) I_{(\delta < \frac{1}{2})} \\ \Big(X_{r_1,G} + \frac{1}{2}(\lambda + 1)\big(X_{r_2,G} + X_{r_3,G} - 2X_{r_1,G}\big)\Big) I_{(\delta \geqslant \frac{1}{2})} \end{cases}$$

where δ is as above. It alternates between differential mutation and three-point recombination.

19.1.4.3.7 Scheme DE7 Alternatively, we have the two possible strategies using an either-or algorithm. If we favor the r_1 event we have

$$V = \Big(X_{r_1,G} + F\big(X_{r_2,G} - X_{r_3,G}\big)\Big) I_{(\delta < P_e)}$$

while if we favor the best event we have

$$V = \Big(X_{best,G} + F\big(X_{r_2,G} - X_{r_3,G}\big)\Big) I_{(\delta < P_e)}$$

It alternates between differential mutation and doing nothing.

19.1.4.3.8 Scheme DE8 Still another more complex algorithm than the Scheme DE1 in Section (19.1.4.3.1) is to introduce two strategies V_l for $l = 1, 2$. We first build the Base vector $\widehat{V}_1$ as the linear combination of the two original base vector $X_{r_1,G}$ and $X_{best,G}$, getting

$$\widehat{V}_1 = \lambda_{M,G} X_{best,G} + \overline{\lambda}_{M,G} X_{r_1,G}$$

where $\overline{\lambda}_{M,G} = 1 - \lambda_{M,G}$. This time $\lambda_{M,G}$ is a Gaussian variable per-generation with mean μ and variance ξ to be defined. The second Base vector $\widehat{V}_2$ is generated according to the rule

$$\widehat{V}_2 = 2X_{best,G} - \widehat{V}_1 = (2 - \lambda_{M,G}) X_{best,G} - \overline{\lambda}_{M,G} X_{r_1,G}$$

The two Trial vectors V_i for $i = 1, 2$ becomes

$$\begin{aligned} V_1 &= \widehat{V}_1 + F_b\big(X_{r_3,G} - X_{r_2,G}\big) \\ V_2 &= \widehat{V}_2 + F_b\big(X_{r_2,G} - X_{r_3,G}\big) = 2X_{best,G} - V_1 \end{aligned}$$

where F_b is a Gaussian variable with mean $\mu = 0$ and variance $\xi = F$. In the special case where $\mu = 2$ and $\xi = 0$ we get

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}\widehat{V}_1 &= 2X_{best,G} - X_{r_1,G} \\ \widehat{V}_2 &= X_{r_1,G}\end{aligned}$$

and the system simplifies to

$$\begin{aligned}V_1 &= 2X_{best,G} - X_{r_1,G} + F_b\big(X_{r_3,G} - X_{r_2,G}\big) \\ V_2 &= X_{r_1,G} + F_b\big(X_{r_2,G} - X_{r_3,G}\big)\end{aligned}$$

and V_2 recover a pseudo Secheme DE1 where $F_b \in [-F, F]$.

19.1.4.4 Improvements

The DE algorithm is found to be a powerful evolutionary algorithm for global optimisation in many real problems. As the DE algorithm performs mutation based on the distribution of the solutions in a given population, search directions and possible step sizes depend on the location of the individuals selected to calculate the mutation values. As a result, since the original article of Storn and Price [1995] many authors improved the DE model to increase the exploration and exploitation capabilities of the DE algorithm when solving optimisation problems. We are going to review a few changes to the DE algorithm which greatly improved the performances of our problem.

19.1.4.4.1 The tuning parameters The tuning of the DE is mainly contolled by three variables: the number of vector (or individual) NP, the differential weight F, and the crossover factor CR. Finding bounds for their values has been a topic of intensive research. Zaharie [2002] studied theoretically the convergence of the DE by analysing the behavioural parameters of the DE models. She proved that the mutation scale factor F should never be smaller than F_{crit} defined as

$$F_{crit} = \sqrt{\frac{(1 - \frac{CR}{2})}{NP}}$$

Price et al. [2005] showed that only high values of CR guarantee the contour matching properties of DE. Further, only when $CR = 1$ is the mean number of function evaluation for an objective function and its rotated counterpart the same. In that setting, the DE is called rotationally invariant. As a rule of thumb (see Storn et al. [1997]), we get

- $F \in [0.5, 1.0]$
- $CR \in [0.8, 1.0]$
- $NP = 10D$

but they lack generality. Hence, the need to compute these parameters automatically.

19.1.4.4.2 Ageing The DE selection is based on local competition only. The number of children that may be produced to compete against the parent $X_{i,G}$ should be chosen sufficiently high so that a sufficient number of child will enter the new population. Otherwise, it would lead to survival of too many old population vectors that may induce stagnation. To prevent the vector $X_{i,G}$ from surviving indefinitely, Storn [1996] used the concept of ageing. One can define how many generations a population vector may survive before it has to be replaced due to excessive age. If the vector $X_{i,G}$ is younger than Num generations it remains unaltered otherwise it is replaced by the vector $X_{r_3,G}$ with $r_3 \neq i$ being a randomly chosen integer in $[0, NP - 1]$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

19.1.4.4.3 Constraints on parameters Commonly, we are searching for a solution to an optimisation problem between certain bounds. Given the parent vector $X_{i,G}$ for $i = 0, .., NP - 1$ we define upper and lower bounds for each initial parameters as

$$L(j) \leqslant X_{i,G_0}(j) \leqslant U(j)\,,\, j = 0, ..., D-1$$

where G_0 is inception, and we randomly select the initial parameter values uniformly on the interval $[L(j), U(j)]$ as

$$X_{i,G_0}(j) = L(j) + U_j(0,1)\big(U(j) - L(j)\big)\,,\, j = 0, ..., D-1$$

where $U_j(0,1)$ generates a random number in the range $[0,1]$ with a uniform distribution for each element j of the vector. Obviously, as the number of generation G increases, the DE algorithm will generate elements of the vector outside of the limits established (lower and upper) by an amount. Several alternatives exist for handling boundary constraints (see Onwubolu [2004]). Following Mezura-Montes et al. [2004a], this amount is substracted or added to the limit violated (reflecting barrier), in order to shift the value inside the limits. Should this cause the new value to violate the other bound, a random value will be generated, as when creating the initial parameter set. We can also set the element of the vector half way between the old position and the limit as follow

$$U_{i,G+1}(j) = \begin{cases} \frac{1}{2}(X_{i,G}(j) + U(j)) \text{ if } U_{i,G+1}(j) > U(j) \\ \frac{1}{2}(X_{i,G}(j) + L(j)) \text{ if } U_{i,G+1}(j) < L(j) \\ U_{i,G+1}(j) \text{ otherwise} \end{cases}$$

where $U_{i,G+1}(j)$ is the new Trial vector.

19.1.4.4.4 Convergence In order to accelerate the convergence process, when a child replaces its parent, Mezura-Montes et al. [2004a] copied its value both into the new generation and into the current generation. It allows the new child, which is a new and better solution, to be selected among the r_l solutions and create better solutions. Therefore, a promising solution does not need to wait for the next generation to share its genetic code. Similarly, to improve performance and to accelerate the convergence process, Storn [1996] explored the idea of allowing a solution to generate more than one offspring. Once a child is better than its parent, the multiple offspring generation ends. Following the same idea, Coello Coello and Mezura-Montes [2003] and then Mezura-Montes et al. [2006] allowed for each parent at each generation to generate $k > 0$ offspring. Among these newly generated solutions, the best of them is selected to compete against its parent, increasing the chances to generate fitter offspring.

19.1.4.4.5 Self-adaptive parameters It was proved that key control parameters in the DE algorithm, such as the crossover CR and the weight applied to random differential F, should be altered in the evolution process itself (see Liu et al. [2002]). These parameters can be adjusted by using heuristic rules, or they can be self-adapted (see Liu et al. [2005], Brest et al. [2006], Balamurugan et al. [2007]). That is, the control parameters are not required to be pre-defined and can change during the evolution process. These control parameters are applied at the individual levels in the population, such that better values should lead to better individuals producing better offspring and hence better values. However, in general, the random change to the parameters are applied irrespective of the quality of the current parameters. Noman et al. [2011] proposed to preserve better parameter choices, while changing the non-productive ones. We first describe the algorithm proposed by Brest et al. [2006] (jDE), and then introduce the adaptative algorithm of Noman et al. [2011] (aDE). The parameter F is a scaling factor controlling the amplification of the difference between two individuals to avoid search stagnation. At generation $G = 1$ the amplification factor $F_{i,G}$ for the ith individual ($i = 0, ..., NP - 1$) is generated randomly in the range $[0.1, 1.0]$. Then, at the next generations the control parameter is given by

$$F_{i,G+1} = \begin{cases} F_L + r_1 \times F_U \text{ if } r_2 < \tau_1 \\ F_{i,G} \text{ otherwise} \end{cases}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where r_j, for $j = 1, 2$ are uniform random values in $[0, 1]$, $F_L = 0.1$, $F_U = 0.9$ and τ_1 represent the probability to adjust the parameter F. Using the notation in Section (19.1.4.3.5), we can rewrite the scaling factor as

$$F_{i,G+1} = \begin{cases} U_i(F_l, F_u) \text{ if } r_2 < \tau_1 \\ F_{i,G} \text{ otherwise} \end{cases}$$

where $F_l = 0.1$ and $F_u = F_L + F_U = 1.0$. The only difference with the dither is the fact that the randomness of the differential weight depends on a probability of adjustment. According to Feoktistiv [2006], at the beginning of the evolution procedure, the mutation step length, and hence $F_{i,G}$, should be large, as individuals are far away from each other and the procedure could benefit from exploring beyond the current solution space. Since the individuals of a generation converge for subsequent generations, the step length, and hence $F_{i,G}$ should become smaller to allow a more concentrated search around the successful solution space. Thus, by applying custom parameters at individual levels, better values in the population lead to better individuals producing better Donor vectors and so on. Similarly, we can extend the crossover parameter CR as follow

$$CR_{i,G+1} = \begin{cases} r_3 \text{ if } r_4 < \tau_2 \\ CR_{i,G} \text{ otherwise} \end{cases}$$

where r_j, for $j = 3, 4$ are uniform random values in $[0, 1]$ and τ_2 represent the probability to adjust the parameter CR. The new parameter CR takes random values in the range $[0, 1]$. Since $F_L = 0.1$, $F_U = 0.9$, Brest et al. [2006] proposed to set $\tau_1 = \tau_2 = 0.1$ such that F takes random values in the range $[0.1, 1]$. Note, both $F_{k,G+1}$ and $CR_{k,G+1}$ are obtained before the mutation is performed in order to influence the mutation, crossover, and selection of the new vector $X_{i,G+1}$. Given that random adjustment can only be good when the parameter setting is not suitable, Noman et al. [2011] proposed to compare the fitness of the offspring $f(U_G)$ with the average fitness value of the current generation, $f_{a,G}$. Then, the choice of the amplification factor F in offspring U_G for the parent $X_{i,G}$ is given by

$$F_G^{child} = \begin{cases} F_{i,G} \text{ if } f(U_G) < f_{a,G} \\ r_1(0.1, 1.0) \text{ otherwise} \end{cases}$$

and that of the crossover parameter is given by

$$CR_G^{child} = \begin{cases} CR_{i,G} \text{ if } f(U_G) < f_{a,G} \\ r_2(0.1, 1.0) \text{ otherwise} \end{cases}$$

where $r_j(a, b)$ are uniform random number in the range $[a, b]$. Initially, the parameters F and CR are created randomly for each individual. Note, the objective vector part of the offspring is created by using the original mutation and crossover values of the DE.

***Remark* 19.1.1** *Again, perturbing behavioural parameters introduce new parameters in the form of the boundaries for the stochastic sampling ranges and the adjustment probabilities.*

19.1.4.4.6 Selection Santana-Quintero et al. [2005] maintained two different populations (primary and secondary) according to some criteria and considered two selection mechanisms that are activated based on the total number of generation G_{max} and the parameter $sel_2 \in [0.2, 1]$ which regulates the selection pressure. That is

$$\text{Type of selection } = \begin{cases} \text{Random if } G < (sel_2 * G_{max}) \\ \text{Elitist otherwise} \end{cases}$$

a random selection is first adopted followed by an elitist selection. In the random selection, three different parents are randomly selected from the primary population while in the elitist selection they are selected from the secondary one. In both selections, a single parent is selected as a reference so that all the parents of the main population will be reference parents only once during the generating process.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

19.1.4.5 Convergence criteria revised

When applying a genetic algorithm to solve some NP-hard optimisation problem (the optimum is unkown), it is usually infeasible to compute the optimal solution of the problem. Still, we need to determine whether or not the algorithm has converged to some optimum. Since a genetic algorithm is said to converge when there is no significant improvement in the values of fitness of the population from one generation to the next, there is no defined difference between stopping criteria and convergence criteria. That is, the stopping criterion provides the user a guideline in stopping the algorithm with an acceptable solution close to the optimal solution. While, mathematically, that closeness may be judged in various ways, termination criteria should avoid needless computation and prevent premature termination. Thus, stopping criteria should account for

- Reliability guarantees termination within a finite time.
- Performace guarantees no premature termination and no needless computation.

Stopping criteria are generally based on time or fitness value. The simpliest termination criteria, called exhaustion-based criteria, is to select a maximum number of generations of evaluation functions, or, a maximal time budget (absolute time, CPU time). Alternatively, we can terminate the search when the best objective value $f^* = f_{min}$ reaches or surpasses a bound, $f^* \leqslant f_{lim}$. Usually, if there is no change in the best fitness value for K consecutive iterations, the algorithm is terminated. This method works well if the optimum or a lower bound is known. Some authors proposed tight bounds on the number of iterations required to achieve a level of confidence to guarantee that a Genetic Algorithm has seen all strings (see Aytug et al. [1996]). Giggs et al. [2006] empirically studied a way to determine the maximum number of generations. However, determining the optimum time, or, finding a lower bound is a challenge. Thus, stopping criteria should contain the advantage of reacting adaptively to the state of the optimisation run.

The criteria based on objective function values use the underlying fitness function values to calculate auxiliary values as a measure of the state of the convergence of the GA. For instance,

- improvement-based criteria monitor the improvement of the best objective function value (ImpBest) (or its average ImpAvg) along the optimisation process, and stops when it falls below a user-defined threshold for a given number of generations.
- movement-based criteria monitor the distances between the population members in successive iterations (see Schwefel [1995]). The movement in the population can be calculated with respect to the average objective function value (MovObj), or with respect to positions (MovPar). Termination occurs when it is below a threshold for a given number of generations.
- distribution-based criteria monitor the distances of the population members at each iteration (see Zielinski et al. [2008]). It is assumed that all individuals converge to the optimum, such that convergence is reached when they are close to each other. MaxDist is when the maximum distance from every vector to the best population vector is below a threshold.
- combined criteria use several criteria in combination.

It is understood that the algorithm often focuses on global optimisation at the beginning, leading to large movements between population members in successive iterations, and that at the final stages of the optimisation process, the population generally converges to one point. There are different ways of measuring these distances, such as the standard deviation of positions to all, or part, of the population members, or the distance between the individuals with the best and worst objective function value. In any case, when the distance falls below a user-defined threshold for a given number of iterations, the algorithm terminates. For example, the Running Mean is the difference between the current best objective value $f_{min}(G)$, at generation G, and the average of the best objective value

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$f_{a,min}(n) = \frac{1}{G_{last}} \sum_{i=0}^{n} f_{min}(G_i) \, , G_i = G - i\Delta G$$

over a period of time $G_{last} = n\Delta G$, where ΔG is one generation. It must be less or equal to a given threshold, that is,

$$|f_{min}(G) - f_{a,min}(n)| \leqslant \epsilon$$

The Best-Worst is the difference between the best objective value $f_{min}(G)$ and the worst one $f_{max}(G)$ at generation G. At least $p\%$ of the individuals must be feasible, and it must be less or equal to a given threshold, $|f_{min}(G) - f_{max}(G)| \leqslant \epsilon$. We can also consider relative termination criterions such as $\frac{f_{min}}{f_{a,G}} \leqslant \epsilon$, or letting d_{ij} be the sum of all normalised distance between all individuals of the current generation, the ratio $\frac{d_{ij}}{K_{max}} \leqslant \epsilon$. The latter values the spacial speading of individuals of the current generation in the search space (normalised Euclidian distances $\frac{d_{ij}}{d}$ where d is the length of diagonal of the search space). Jain et al. [2001] proposed the Clus Term, combining information from the objective values and the distribution of individuals in the search space. They perform a cluster analysis (single linkage method) of the fittest individuals and determine the total amount $N(t)$ of of individuals in clusters. The search is terminated when the change of the average of $N(t)$ is equal or less than ϵ. Analysis of stopping criteria reacting adaptively to the state of an optimisation were perfomed (see Zielinski et al. [2005], Zielinski et al. [2007]).

Generally, EAs are stopped or terminated when the variance of fitness values of all the strings in the current population is less than a predefined threshold ϵ. It is assumed that after significantly many iterations the fitness values of the strings present in the population are all close to each other (the population becomes homogeneous), thereby making the variance of the fitness values close to 0. Thus ϵ should be chosen close to zero. Further, one should select a significant number of iterations from which the fitness values will be considered in calculating the variance so that the algorithm gets enough opportunity to yield improved (better) solution. However, this is not correct since

- in elitist model, or other GAs, only the best string is preserved.
- any population containing an optimal string is sufficient for the convergence of the algorithm.
- there is a positive probability of obtaining a population after infinitely many iterations with exactly one optimal string and others are being not optimal.

A lot more iterations often occur after the global optimum has been reached, or nearly reached, to improve other inferior individuals. However, only the best objective function value is important, rather than the convergence of the whole population. Since at the beginning, global search dominates the optimisation algorithm, while at final stages, the algorithm focuses on local optimisation, Liu et al. [2009] proposed a combination of global and local methods. They monitor the average improvement of the whole population in the former, and they monitor the best objective function value in the latter. Bhandari et al. [2012] established theoretically that the variance of the best fitness values obtained in the iterations can be considered as a measure to decide the termination criterion of a GA with elitist model (EGA). The proposed criterion uses only the fitness function values and takes into account the inherent properties of the objective function. We let $f_{min}(i)$ be the best fitness function value obtained at the end of ith iteration, such that $f_{min}(1) \geqslant f_{min}(2) \geqslant ... \geqslant f_{min}(n) \geqslant ... \geqslant F_1$, where F_1 is the global optimal value of the fitness function (F_i denotes the ith lowest fitness function value). Then we get the statistical mean and variance of the best fitness values obtained up to the nth iteration as

$$\begin{aligned} f_{min,1}(n) &= \frac{1}{n} \sum_{i=1}^{n} f_{min}(i) \\ b(n) &= Var(f_{min}(n)) = \frac{1}{n} \sum_{i=1}^{n} \left(f_{min}(i) - f_{min,1}(n)\right)^2 = f_{min,2}(n) - f_{min,1}^2(n) \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $f_{min,2}(n) = \frac{1}{n}\sum_{i=1}^{n} f_{min}^2(i)$. The variance can be iteratively calculated as follow

$$b_{n+1} = \frac{1}{n+1}\Big((nf_{min,2}(n) + f_{min}^2(n+1)) - \big(f_{min,1}(n) + f_{min}(n+1)\big)^2\Big)$$

such that only $f_{min,1}(n)$ and $f_{min,2}(n)$ at step n are required to keep in memory when computing the variance at step $(n+1)$. Alternatively, following Equation (24.3.17), we can write recursively the sample mean as

$$f_{min,1}(n+1) = f_{min,1}(n) + \frac{f_{min}(n+1) - f_{min,1}(n)}{n+1}$$

and following Equation (24.3.18), the sample variance becomes

$$b(n+1) = b(n) + f_{min,1}^2(n) - f_{min,1}^2(n+1) + \frac{f_{min}^2(n+1) - b(n) - f_{min,1}^2(n)}{n+1}$$

So far, the variance based criterion is not scale invariant, meaning it is sensitive to transformations of the fitness function [1]. One can easily avoid the impact of the scaling effect by a simple transformation of the fitness function, such as

$$g(x) = \frac{f(x)}{f_{min}^1}$$

where f_{min}^1 is the minimum value of the fitness function obtained in the first iteration. If we let $b_n(g)$ be the variance of the best fitness values obtained up to the nth iteration for the function $g(x)$, then we get

$$b_n(g) = \frac{1}{n(f_{min}^1)^2}\sum_{i=1}^{n}\big(f_{min}(i) - f_{min,1}(n)\big)^2 = \frac{1}{(f_{min}^1)^2}b_n(f)$$

such that assuming the tolerance level ϵ_f as the value ϵ for the function f is equivalent to assuming $\epsilon_f \times (f_{min}^1)^2$ as the value of ϵ for the function g. It implies that the user has to adjust the value of ϵ for the applied transformation. Note, we now need to use Equation (24.3.19) to obtain recursively the sample mean, and Equation (24.3.20) to obtain recursively the sample variance. In that setting, the GA is stopped or terminated after N iterations when the variance of the best fitness values obtained so far is bounded. That is, $b_N < \epsilon$, where $\epsilon > 0$ is a user defined small quantity corresponding to the difference between the fitness value of the best solution obtained so far and the global optimal solution.

19.2 Handling the constraints

19.2.1 Describing the problem

We saw above that EAs in general, and DE in particular, lacked a mechanism to deal with the constraints of the problems. Recently, various academics worked on solving that problem, and one of the most popular constraint handling mechanisms was proposed by Deb [2000] on genetic algorithms (GAs) who used the three feasibility rules. This algorithm generates feasible individuals, while maintaining a reasonable ratio between feasible and infeasible members in a population, allowing for a sparse feasible domain. Several studies assessed the performances of these rules on a large number of test functions, and very good results were found (see Zielinski et al. [2006], Zhang et al. [2012]). Before reviewing his method and showing how it was improved, we recall that Michalewicz [1995] discussed different constraint handling methods used in GAs and classified them in five categories

- methods based on preserving feasibility of solutions, that is, we use a search operator that maintains the feasibility of solutions

[1] $g(x) = k \times f(x)$ where k is a constant.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- methods based on penalty functions
- methods making distinction between feasible and infeasible solutions using different search operators for handling infeasible and feasible solutions
- methods based on decoders using an indirect representation scheme which carries instructions for constructing feasible solutions
- hybrid methods where evolutionary methods are combined with heuristic rules or classical constrained search methods

In a single-objective optimisation problem, the traditional approach for handling constraints is the penalty function method. The fitness of a candidate is based on a scale function F which is a weighted sum of the objective function value and the amount of design constraint violation

$$F(X) = f_1(X) + \Big(\sum_{k=1}^{p} \omega_k \max\left(g_k(X), 0\right) + \sum_{k=p+1}^{q} \omega_k |h_k(X)|\Big)$$

where ω_k are positive penalty function coefficients and such that the kth constraint $g_k(\cdot)$ and $h_k(\cdot)$ should be normalised. This method requires a careful tuning of the coefficients ω_k to obtain satisfactory design, that is a balance between the objective function and the constraints but also between the constraints themselves. Kusakci et al. [2012] presented a literature review summerising the constraint handling techniques for constrained optimisation problems (COPs).

19.2.2 Defining the feasibility rules

To overcome this problem, Deb [2000] proposed a penalty function approach based on the non-dominance concept, ranking candidates using the definition of domination between two candidates.

Definition 19.2.1 *A solution i is said to dominate a solution j if both of the following conditions are true*

1. *solution i is no worse than solution j in all objective*

$$\forall f_m(X_i) \leqslant f_m(X_j)$$

2. *solution i is strictly better than solution j in at least one objective*

$$\exists f_m(X_i) < f_m(X_j)$$

The constrained domination approach ranks candidates according to the following definition

Definition 19.2.2 *A solution i is said to constrained-dominate a solution j if any of the following conditions is true*

1. *solutions i and j are feasible and solution i dominates solution j.*
2. *solution i is feasible and solution j is not.*
3. *both solutions i and j are infeasible but solution i has a smaller constraint violation.*

He let the fitness function be

$$F(X) = \begin{cases} f(X) \text{ if } g_k(X) \leqslant 0 \ \forall k = 1, 2, .. \\ f_{max} + TACV \text{ otherwise} \end{cases}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where f_{max} is the objective value with the worst feasible solution in the population and (TACV) is the total amount of constraint violation

$$TACV = \sum_{k=1}^{p+q} \max\left(g_k(X), 0\right)$$

Therefore, solutions are never directly compared in terms of both objective function and constraint violation information. However, the high selection pressure generated by tournament selection will induce the use of additional procedure to preserve diversity in the population such as niching or sharing. Clearly, there is no tuning of the penalty function coefficients when the number of constraint is one. But, when multiple constraints are considered some considerations must be taken to relate constraints together. One way forward is to normalise the constraints such that every constraint has the same contribution to the comparing value as was done by Landa Becerra et al. [2006]. Letting $g_{max}(k)$ be the largest violation of the constraint $\max\left(g_k(X), 0\right)$ found so far, we define the new TACV as

$$NTACV = \sum_{k=1}^{p} \frac{\max\left(g_k(X), 0\right)}{g_{max}(k)}$$

19.2.3 Improving the feasibility rules

Again, many different approaches were proposed, for instance Coello Coello [2000] modified the definition of the constrained domination approach given in Definition (19.2.2) such that if the individuals are infeasible he compares the number of constraints violated first and only in the case of a tie would he use the total amount of constraint violation in the definition, getting

Definition 19.2.3 *A solution i is said to constrained-dominate a solution j if any of the following conditions is true*

1. *solutions i and j are feasible and solution i dominates solution j.*
2. *solution i is feasible and solution j is not.*
3. *both solutions i and j are infeasible but solution i violates less number of constraints than solution j.*
4. *both solutions i and j are infeasible and violating the same number of constraints but solution i has a smaller TACV than solution j.*

In that setting, the fitness of an infeasible solution not only depends the amount of constraint violation, but also on the population of solutions at hand. However, this technique may not be very efficient when the degrees of violation of constraints $g_k(X)$ are significantly different because the TACV is a single value. Alternatively, Coello Coello et al. [2002] handled constraints as additional objective functions and used the non-dominance concept (on objective functions) in Definition (19.2.1) to rank candidates. As a result, it required solving the objective function a large number of time. Going one step further, Oyama et al. [2005] introduced dominance in constraint space.

Definition 19.2.4 *A solution i is said to dominate a solution j in constraint space if both of the following conditions are true*

1. *solution i is no worse than solution j in all constraints*

$$\forall g_k(X_i) \leqslant g_k(X_j)$$

2. *solution i is strictly better than solution j in at least one constraint*

$$\exists g_k(X_i) < g_k(X_j)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Introducing the idea of non-dominance concept to the constraint function space, their proposed constraint-handling method is

Definition 19.2.5 *A solution i is said to constrained-dominate a solution j if any of the following conditions is true*

1. *solutions i and j are feasible and solution i dominates solution j in objective function space.*
2. *solution i is feasible and solution j is not.*
3. *both solutions i and j are infeasible but solution i dominates solution j in constraint space.*

In that setting, any non-dominance ranking can be applied to feasible designs and infeasible designs separately. As a result, in a single-objective constrained optimisation problem, Bloch [2010] modified the dominance-based tournament selection of Coello and Mezura with the non-dominance concept of Oyama et al., getting

Definition 19.2.6 *The new dominance-based tournament selection is*

1. *if solutions i and j are both feasible and solution i dominates in objective function solution j then solution i wins.*
2. *if solution i is feasible and solution j is not, solution i wins.*
3. *if solutions i and j are both infeasible and solution i dominates in constraint space solution j then solution i wins.*
4. *if solutions i and j are infeasible and non-dominated in constraint space, if solution i violates less number of constraints than solution j then solution i wins.*
5. *if solutions i and j are both infeasible, non-dominated in constraint space and violating the same number of constraints but solution i has a smaller TACV than solution j then solution i wins.*

19.2.4 Handling diversity

In order to explore new regions of the search space and to avoid premature convergence, a set of feasibility rules coupled with a diversity mechanism is proposed by Mezura-Montes et al. [2006]. It maintains besides competitive feasible solutions some solutions with a promising objective function value allowing for the DE to reach optimum solutions located in the boundary of the feasible region of the search space. Given a parameter S_r one can choose to select between parent and child based either only on the objective function value or on feasibility. As more exploration of the search space is required at the beginning of the process, the parameter S_r will be decreasing in a linear fashion with respect to the number of generation. Given an initial value $S_{r,0}$ and a terminal value $S_{r,\infty}$, the adjustment of the parameter is

$$S_{r,G+1} = \begin{cases} S_{r,G} - \Delta_{S_r} \text{ if } G > G_{max} \\ S_{r,\infty} \text{ otherwise} \end{cases}$$

where $\Delta_{S_r} = \frac{S_{r,0} - S_{r,\infty}}{G_{max}}$. After some generations, assuming that promising areas of the feasible region have been reached, we focus on keeping the feasible solutions found discarding the infeasible ones. In that setting, infeasible solutions with good objective function values will have a significant probability of being selected which will slowly decrease as the number of generation increases.

In some problems, it does not make sense to use the infeasible individual unless they are very close to the boundary between the feasible and infeasible region. This is the case of the calibration problem given in Section (1) where the constraints ensure that prices do not violate the AAO rules. To circumvent this issue, Mezura-Montes et al. [2004a] proposed that at each generation the best infeasible solution with lowest amount of constraint violation both in the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

parents (μ) and the children (λ) population will survive. Either of them being chosen with an appropriate probability, it will allows for infeasible solutions close to the boundary to recombine with feasible solutions. The pseudo code for introducing diversity in the DE algorithm is

```
if  flip(S_r)  then
   Select the best infeasible individual from the children population
else
   Select the best individual based on five selection criteria
end if
```

19.2.5 The proposed algorithm

Using the improvements in Section (19.1.4.4) and applying the Definition (19.2.6) on dominance-based tournament selection to handle the constraints, the pseudo code for the DE algorithm with constraints becomes

```
Begin
 G = 0  and  Age_i,G  =  0 ∀ i,  i = 1,..,NP
Create a random initial population  X_i,G ∀ i,  i = 1,..,NP
Evaluate  f(X_i,G) ∀ i,  i = 1,..,NP
while  niter  <  max_iter  and  G  <  Gmax  do
    fmin_old  =  fmin
   for  i = 1  to  NP  do
      for  k = 1  to  N_K  do
         Select randomly  r_1 ≠ r_2 ≠ r_3 ≠ i
          j_r = U(1,N)
         for  j = 1  to  N  do
            if  U(0,1)  <  CR  or  j = j_r  then
                U_i,G(j)  =  X_r_3,G(j)  +  F (X_r_1,G(j)  −  X_r_2,G(j))
            else
                U_i,G(j)  =  X_i,G(j)
            end if
         end for
         if  k  >  1  then
            if  U_i,G(j)  is better than  U_i_best,G(j)  based on five selection
            criteria then
                U_i_best,G(j)  =  U_i,G(j)
         else
             U_i_best,G(j)  =  U_i,G(j)
      end of for
      Apply diversity :
      if  flip(S_r)  then
         if  f(U_i_best,G)  ⩽  f(X_i,G)  then
             X_i,G+1  =  X_i,G  =  U_i_best,G
         else
             X_i,G+1  =  X_i,G
         end if
      else
         if  U_i_best,G  is better than  X_i,G  based on five selection criteria then
             X_i,G+1  =  X_i,G  =  U_i_best,G
         else
```


This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

```
            if  Age_i,G < N_A or i = i_best then
                X_i,G+1 = X_i,G
            else
               Select randomly r_4 ≠ i
                X_i,G+1 = X_r_4,G
                Age_i,G = 0
            end if
         end if
      end if
      if  fmin > f(X_i,G)
          fmin = f(X_i,G)
         i_best = i
      end if
   end for
   Apply convergence criterions:
   if  fmin_old − f_min < precision
      niter = niter + 1
   else
      niter = 0
   end if
    G = G + 1
end while
End
```

19.3 Genetic programming

We follow the books by Koza [1992] [1994], the survey by Langdon et al. [1995], and the report by Poli et al. [2007].

19.3.1 An overview

Genetic programming (GP) is an evolutionary computation (EC) technique that automatically solves problems without having to tell the computer explicitly how to do it. At the most abstract level GP is a systematic, domain-independent method for getting computers to automatically solve problems starting from a high-level statement of what needs to be done. The term genetic programming was suggested by David Goldberg to describe the automatic production of programs by using a hierarchical genetic algorithm to search the space of possible programs.

The goal of having computers automatically solve problems is central to artificial intelligence, machine learning, and the broad area encompassed by what Turing called machine intelligence (see Alan Turing [1950]). Richard Forsyth [1981] demonstrated the successful evolution of small programs (BEAGLE), represented as trees, to perform classification of crime scene evidence for the UK Home Office. The idea of evolving programs was current amongst John Holland's students. Proceedings of the International Conference on Genetic Algorithms and the Applications at Carnegie Mellon University, Cramer [1985] published evolved programs in two specially designed languages, LISP and Scheme. This is because statements in LISP can easily be converted to a parse tree representation, and evaluation involves compilation and provision of data. John Koza (also a PhD student of John Holland) [1990] patented his invention of a GA for program evolution, which was followed by publication in the International Joint Conference on Artificial Intelligence (see Koza [1989]). It uses genetic operators that modify a set of LISP statements. It was followed by over 205 publications and a series of four books (see Koza [1992], [1994]). The annual Genetic Programming conference, which was followed in 1998 by the annual EuroGP conference.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

19.3.2 Description

Genetic programming (GP) is a technique whereby computer programs (individuals in GA) are encoded as a set of genes that are then modified (evolved) using an evolutionary algorithm (EA). Following the process described in Section (19.1.1.3), the population is composed of chromosomes, or parse trees (see Section (4.1.2)), where each chromosome of the population is evaluated with a fitness function. Thus, generation by generation GP iteratively transforms populations of programs into other populations of programs. During the process, GP constructs new programs by applying genetic operations which are specialised to act on computer programs. That is, new programs are produced by removing branches from one tree and inserting them into another one. This process ensures that the programs remain trees and stay syntactically valid. Thus, genetic programming is fundamentally different from simply shuffling lines of machine code. Formally, the sequence of operations are:

- Crossover: the creation of one or two offspring programs by recombining randomly chosen parts from two selected programs.
- Mutation: the creation of one new offspring program by randomly altering a randomly chosen part of one selected program.
- Reproduction: it copies selected individuals in the next generation. It is typically applied only to produce a fraction of the new generation.

More generally, GP may be understood as a symbolic regression, where the model function is not suggested by the user but constructed by the algorithm. It comprises five preliminary steps to solving a problem which are:

1. choosing the terminals,
2. the functions,
3. the fitness function,
4. the control parameters, and
5. the termination criterion.

See implementation in Algorithm (64).

Algorithm 64 Abstract GP Algorithm

```
Randomly create an initial population of programs from the available primitives.
repeat
    Execute each program and ascertain its fitness.
    Select one or two program(s) from the population with a probability based on fitness to participate in genetic
    operations.
    Create new individual program(s) by applying genetic operations with specified probabilities.
until an acceptable solution is found or some other stopping condition is met.
return the best so-far individual.
```

In some cases, we can include architecture-altering operations which randomly alter the architecture (for instance, the number of subroutines) of a program to create a new offspring program.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

19.3.2.1 Representation, initialisation and operators

19.3.2.1.1 Representation Programs are usually expressed as syntax trees rather than as lines of code. For instance, the tree representation of the program $\max\left(x^2, x+3y\right)$ is given in Figure (19.13). Using a prefix notation similar to that used in LISP or Scheme, this expression becomes $\left(\max(*xx)(+x(*3y))\right)$.

Figure 19.13: GP syntax tree for $\max\left(x^2, x+3y\right)$

In this example, we get

- The terminals (variables and constants): are leaves of the tree.
- The internal nodes (functions): the arithmetic operations (addition, subtraction, multiplication, and division). The connections between the terminals and functions indicate the order in which operations are to be performed.
- The primitive set: the sets of all functions and terminals.

When the programs are composed of multiple components (subroutines), the representation used in GP is a set of trees (one for each component) grouped together under a special root node acting as glue. The sub-trees are called branches. These branches and other features of the structure form the architecture of the program (see Figure (19.14)).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 19.14: GP architecture

19.3.2.1.2 Initialisation Some methods that randomly initialise the population are:

- The full method: the initial individuals are generated subject to a pre-established maximum depth. The nodes are taken randomly from the function set until the maximum tree depth is reached. Beyond that depth only terminals can be chosen.
- The grow method: it allows for the creation of trees of varying size and shape. Here nodes are selected from the whole primitive set until the depth limit is reached, below which only terminals may be chosen (as is the case in the Full method). The size and shapes of the trees generated this way are highly sensitive to the sizes of the function and terminal sets.
- Ramped half-and-half: it is a combination of the two above methods (see Koza [1992]). Half the initial population is constructed using Full and half is constructed using Grow. This is done using a range of depth limits (hence the term ramped) to help ensure that we generate trees having a variety of sizes and shapes.

19.3.2.1.3 Operators These often make it difficult to control the statistical distributions of important properties such as the sizes and shapes of the generated trees. In order to get closer control of these properties, other initialisation mechanisms were proposed. Note, the initial population need not be entirely random. If something is known about likely properties of the desired solution, trees having these properties can be used to seed the initial population.
The diversity of the population should be monitored to ensure that there is significant mixing of different initial trees. Techniques such as multi-objective GP, demes, fitness sharing and the use of multiple seed trees, might be used.

19.3.2.2 Selection, recombination and mutation

19.3.2.2.1 Selection The genetic operators are applied to individuals that are probabilistically selected based on fitness (see Section (19.1.3.2)). Some methods commonly employed for selecting individuals are:

- tournament selection
- fitness-proportionate selection
- stochastic universal reward-based selection

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- truncation selection

Any standard EA selection mechanism can be used. However, departure from EAs is in the implementation of crossover and mutation operators.

19.3.2.2.2 Crossover Some of the commonly used method for crossover are:

- Sub-tree crossover: Given two parents, it randomly selects a crossover point in each parent tree and then creates the offspring by replacing the sub-tree rooted at the crossover point in a copy of the first parent with a copy of the sub-tree rooted at the crossover point in the second parent. See Figure (19.15). In general, crossover points are not selected with uniform probability. Typical GP primitive sets lead to trees with an average branching factor of at least 2, so the majority of the nodes will be leaves. As a result, the uniform selection of crossover points leads to crossover operations frequently exchanging only very small amounts of genetic material. To go around, Koza considered the approach of choosing functions 90% of the time, while leaves are selected 10% of the time.
- One-point crossover: It selects a common crossover point in the parent programs and then swaps the corresponding sub-trees. To account for the possible structural diversity of the two parents, one-point crossover analyses the two trees from the root nodes and considers for the selection of the crossover point only the parts of the two trees, called the common region, which have the same topology (the same arity [2] in the nodes encountered traversing the trees from the root node) (see Poli et al. [1998]).
- Context-preserving crossover: The crossover points are constrained to have the same coordinates, like in one-point crossover. However, in this case no other constraint is imposed on their selection (they are not limited to the common region) (see D'haeseleer [1994]).
- Size-fair crossover: The first crossover point is selected randomly like in standard crossover. Then the size of the sub-tree to be excised from the first parent is calculated. This is used to constrain the choice of the second crossover point so as to guarantee that the sub-tree excised from the second parent will not be unfairly big (see Langdon [2000]).

Figure 19.15: GP sub-tree crossover

[2] In logic, mathematics, and computer science, the arity of a function or operation is the number of arguments or operands that the function takes.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, the notion of common region is related to the notion of homology [3], that is, it represents the result of a matching process between parent trees. See details in Figure (19.22). Thus, within such a region, transfer of homologous primitives can happen in a similar fashion as in GAs operating on linear chromosomes.

19.3.2.2.3 Mutation Some of the commonly used method for mutation are:

- Sub-tree mutation: It randomly selects a mutation point in a tree and substitutes the sub-tree rooted there with a randomly generated sub-tree. See Figure (19.16) It is sometimes implemented as crossover between a program and a newly generated random program. This operation is also known as headless chicken crossover.
- Point mutation: It is equivalent for GP of the bit-flip mutation used in GAs. A random node is selected and the primitive stored there is replaced with a different random primitive of the same arity taken from the primitive set. If no other primitives with that arity exist, nothing happens to that node (but other nodes may still be mutated). Note that, when sub-tree mutation is applied, this involves the modification of exactly one sub-tree. Point mutation, on the other hand, is typically applied with a given mutation rate on a per-node basis, allowing multiple nodes to be mutated independently.

Figure 19.16: GP sub-tree mutation

There are a number of mutation operators which treat constants in the program as special cases. For instance, Schoenauer [1996] mutates constants by adding Normally distributed random noise to them.

19.3.3 Running a GP

In order to run a GP system for solving a problem, we need to specify a small number of ingredients, often termed preparatory steps. These steps are:

1. the terminal set
2. the function set

[3] In biology, homology is the existence of shared ancestry between a pair of structures, or genes, in different taxa.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3. the fitness measure
4. certain parameters for controlling the run
5. the termination criterion and method for designating the result of the run

We are now going to present these steps.

19.3.3.1 The terminal set

When evolving programs in GP, the first two preparatory steps

1. the definition of the terminal sets
2. the definition of the function sets

specify a domain-specific language: the ingredients that are available to GP to create computer programs.
The terminal set may consist of:

- the program's external inputs, typically taking the form of named variables (such as x and y).
- functions with no arguments, which are, therefore, interesting either because they return different values in different invocations (for example the function rand() that returns random numbers) , or because the produce side effect (for example go-left()).
- constants, which can either be pre-specified or randomly generated as part of the tree creation process.

19.3.3.2 The function set

The function set is typically driven by the nature of the problem domain. In a simple numeric problem, the function set may consist of the four arithmetic functions. Table (19.17) shows a sample of some of the functions one sees in the GP literature. In many problems, the primitive set includes specialised functions and terminals which are expressly designed to solve problems in a specific domain of application.

Function Set	
Kind of Primitive	*Example(s)*
Arithmetic	`+`, `*`, `/`
Mathematical	`sin`, `cos`, `exp`
Boolean	`AND`, `OR`, `NOT`
Conditional	`IF-THEN-ELSE`
Looping	`FOR`, `REPEAT`
⋮	⋮

Terminal Set	
Kind of Primitive	*Example(s)*
Variables	`x`, `y`
Constant values	`3`, `0.45`
0-arity functions	`rand`, `go_left`

Figure 19.17: GP primitives function and terminal sets

The functions in the set are required to satisfy some properties:

1. Closure: For GP to work efficiently, most function sets are required to have an important property known as closure, which can in turn be broken down into the properties of type consistency and evaluation safety.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Type consistency: It is required since sub-tree crossover can mix and join nodes quite arbitrarily during the evolutionary process. Hence, it is necessary that any sub-tree can be used in any of the argument positions for every function in the function set, since it is always possible that sub-tree crossover will generate that combination. Note, simple restructuring of the functions can resolve apparent problems. We can also explicitly include type information, and constrain operations like crossover so they do not perform illegal (from the standpoint of the type system) operations. The latter is necessitated by the fact that many commonly used functions can fail in various ways. An evolved expression might, for example, divide by 0, or call MOVE FORWARD when facing a wall or precipice. We can modify the standard behaviour of primitives. For instance, use protected versions of numeric functions that can throw exceptions. The protected version of such a function first tests for potential problems with its input(s) before executing the corresponding instruction, and if a problem is spotted some prefixed value is returned.
- Sufficiency: It requires that the primitives in the primitive set are capable of expressing the solutions to the problem, that is, that the set of all the possible recursive compositions of such primitives includes at least one solution. However, it can only be guaranteed for some problems, when theory or experience with other methods tells us that a solution can be obtained by combining the elements of the primitive set. When a primitive set is insufficient for a particular application, GP can only develop programs that approximate the desired one, although perhaps very closely.

2. Evolving structures other than programs: When considering problems in the real world, the primitive set is designed in such a way that, through their execution, programs construct solutions to the problem. It is viewed as analogous to the development by which an egg grows into an adult.

19.3.3.3 The fitness measure

The search space explored in GP includes all the programs that can be constructed by composing the primitives in all possible ways. The task of the fitness measure, effectively (albeit implicitly) specifies the desired goal of the search process. It is the primary (and often sole) mechanism for giving a high-level statement of the problem's requirements to the GP system. Depending on the problem at hand, fitness can be measured in terms of

- the amount of error between its output and the desired output
- the amount of time (fuel, money, etc.) required to bring a system to a desired target state
- the accuracy of the program in recognising patterns or classifying objects into classes
- the payoff that a game-playing program produces
- the compliance of a structure with user-specified design criteria
- other criteria

Since the structures being evolved in GP are computer programs, fitness evaluation normally requires executing all the programs in the population, typically multiple times. However, this is time consuming. One solution is to use an interpreter to evaluate the evolved programs.
Interpreting a program tree means executing the nodes in the tree in an order that guarantees that nodes are not executed before the value of their arguments (if any) is known. This is done by traversing the tree recursively starting from the root node, and postponing the evaluation of each node until the value of its children (arguments) is known. For example, in Figure (19.18), the number to the right of each internal node represents the result of evaluating the sub-tree root at that node. In this example, the independent variable x evaluates to -1. A pseudo-code implementation of the interpretation procedure is given in Algorithm (65).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 19.18: GP interpretation syntax tree

Algorithm 65 Technical Interpreter for GP

Require: $expr$ is an expression in prefix notation, $expr(1)$ is the primitive at the root of the expression, $expr(2)$ represents the first argument of that primitive, $expr(3)$ represents the second argument, etc.

```
if expr is a list then
    proc = expr(1) (Non-terminal: extract root)
    if proc is a function then
        value = proc(eval(expr(2)), eval(expr(3)), ...) (Function: evaluate arguments)
    else
        value = proc(expr(2), expr(3), ...) (Macro: do not evaluate arguments)
    end if
else
    if expr is a variable or expr is a constant then
        value = expr (Terminal variable or constant: just read the value)
    else
        value = expr() (Terminal 0-arity function: execute)
    end if
end if
return value
```

In some problems we might be interested in the output produced by a program, that is, the value returned when we evaluate starting at the root node. We could also be interested in the actions performed by a program. In this case the primitive set will include functions with side effects, that is, functions that do more than just return a value, (for example, change some global data structures). In general, the fitness of a program depends on the results produced by its execution on many different inputs or under a variety of different conditions. They contribute to the fitness value of a program and are called fitness cases. The fitness measures can be multi-objective, that is, they combine two or more different elements that are often in competition with one another (see Deb [2001]).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

19.3.3.4 The parameters and termination

We need to specify the control parameters for the GP run. Some of these parameters are:

- The population size.
- The probabilities of performing the genetic operations. For example, the rate of crossover.
- The maximum size for programs.
- Other details of the run.

We also need to specifying the termination criterion and the method of designating the result of the run. The termination criterion may include a maximum number of generations to be run as well as a problem-specific success predicate. Typically the single best-so-far individual is then harvested and designated as the result of the run, although one might wish to return additional individuals and data as necessary or appropriate for your problem domain.

19.3.4 Advanced techniques

The scope for research into the theory and techniques used in GP is illustrated in Figure (19.19). We are now going to discuss some of these advanced techniques (see Poli et al. [2007] for more details).

Figure 19.19: GP techniques

19.3.4.1 Abstraction

The advantages of abstraction are that it allows the control of complexity and facilitates code reuse. Reuse programs consist of organised sequences of repeated steps into reusable components such as subroutines, functions, and classes. Code reuse is particularly useful for GP because it means that a block of useful code need not be independently rediscovered at different places within the program where it is needed. Thus, we can exploit a problem's modularities, symmetries, and regularities and thereby potentially accelerate the problem-solving process. Programmers can further organise these components into hierarchies in which top level components call lower level ones, which call still lower levels, etc.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

19.3.4.1.1 Automatically defined functions The Automatically Defined Functions (ADFs) (see Koza [1994]) have been very successful at evolving reusable components. Typically, each ADF is a separate tree. A program consists of one (or more) function-defining trees as well as one or more main result-producing trees. It may have none, one, or more inputs. The body of an ADF contains its work-performing steps. Each ADF belongs to a particular program in the population. It may be called by the program's main result-producing tree, by another ADF, or by another type of tree (such as the other types of automatically evolved program components described). The ADFs are called with different inputs. The work-performing steps of the program's main result-producing tree and the work-performing steps of each ADF are automatically and simultaneously created by GP. They have different function and terminal sets.
Koza et al. [1999] proposed other types of automatically evolved program components to reuse code such as automatically defined iterations (ADIs), automatically defined loops (ADLs) and automatically defined recursions (ADRs). Note, automatically defined stores (ADSs) provide means to reuse the result of executing code.

19.3.4.1.2 Abstract data types Langdon [1995] used GP with index memory to successfully evolve queue and stack abstract data types (ADTs) using a chromosome structure with multiple trees. Each tree in the chromosome was used to represent one of the required ADT member functions. The results suggest that it may be possible to extend the level of abstraction in GP from pure functional abstraction as in ADFs, MAs and adaptive representations to full ADTs. This may help increase the scalability of GP.

19.3.4.2 Breeding policies

A number of choices are determined by a breeding policy. These include the choice of parents, the number of offspring produced, the choice of which individuals in the population are to be displaced by newly created individuals.

19.3.4.2.1 Population enrichment Perry [1994] separated the evolution of a GP into enrichment and consolidated phases. As an example, he considered an enrichment phase involving 40 runs with a population size of 2000 individuals, each processed for only 3 generations. The best of each of the 40 runs (the enriched population) were then loaded into a consolidated run with 1960 members of population generated at random. This GP population was then processed for 50 generations. The results indicate that one run of an enriched population produces better results than 3 traditional runs. In addition fewer individuals are processed, the solutions are more robust and parsimonious.

19.3.4.2.2 Tournament selection In GA systems, individuals are chosen for reproduction from the population probablistically in proportion to their fitness. In tournament selection, a number of individuals are selected at random (dependent on the tournament size, typically 7) from the population and the individual with the best fitness is chosen for reproduction. Reverse tournament selection is sometimes used in steady state GP where, the individual with the worst fitness is chosen to be replaced by a newly reproduced individual. The tournament selection approach allows a tradeoff to be made between exploration and exploitation of the gene pool.

19.3.4.2.3 Steady state GP In GP the population size is kept constant, so that when reproduction takes place, new individuals must replace existing individuals. If the number of individuals created is equal to the population size, then the entire population is replaced and an entire new generation is created. This approach is referred to as generational GP. If on the other hand, the number of individuals replaced is actually quite small, for example one, then the new population is actually a mix of new and old generations. This approach is called steady state GP (SSGP). The term generation equivalent is often used in SSGP and refers to the point at which an entire population has been created.

19.3.4.2.4 Elitism Elitism refers to a breeding policies which try to preserve the best individuals in the population. The implication is that it is possible for highly fit individuals to never be replaced. SSGP is an elitist breeding policy since only the individuals with poor fitness are replaced by new individuals. Elitism is also sometimes implemented in generational GP.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

19.3.4.2.5 Demes The standard GP breeding policy is described as panmictic since it potentially allows any individual in the population to breed with any other individual. An alternative is to divide the population into subpopulations called demes, or islands. These demes are typically positioned on a two dimensional grid, and breeding is restricted so that any two parents have a much higher probability of being selected from within the same deme or from geographically close demes.
One of the advantages of the demic approach is that it is easy to implement using parallel or distributed architectures. Koza et al. [1995] developed a next generation GP system using a network of transputers to exploit the parallelism inherent in the demic model. The results of his experiments with EVEN-N-PARITY problems indicated that more than linear speed-up in solving the problem using GP was obtained.

19.3.4.3 Program architecture

The architecture of a program consists of the total number of trees, the type of each tree (result-producing tree, ADF, ADI, ADL, ADR, or ADS), the number of arguments (if any) possessed by each tree, and, if there is more than one tree, the nature of the hierarchical references (if any) allowed among the tree.
There are three ways to determine the architecture of the computer programs that will be evolved:

1. We can specify in advance the architecture of the overall program, that is, perform an architecture-defining preparatory step in addition to the five itemised in Section (19.3.2).

2. The run may employ evolutionary selection of the architecture (as described in Koza [1994]), thereby enabling the architecture of the overall program to emerge from a competitive process during the run of GP.

3. The run may employ a set of architecture-altering operations which can create new ADFs, remove ADFs, and increase or decrease the number of inputs an ADF has. Initially, many architecture changes are designed not to change the semantics of the program and, so, the altered program often has exactly the same fitness as its parent. However, the new arrangement of ADFs may make it easier for subsequent changes to evolve better programs later.

19.3.4.4 Problem solver

The GP Problem Solver (GPPS) is a very powerful AI approach, but typically requires considerable computational time (see Koza et al. [1999]). In that setting, the user does not need to chose either the terminal set or the function set. The function set for GPPS is the four basic arithmetic functions (addition, subtraction, multiplication, and division) and a conditional operator IF. The terminal set for GPPS consists of numerical constants and a set of input terminals that are presented in the form of a vector. By employing this generic function set and terminal set, GPPS reduces the number of preparatory steps from five to three. It relies on the architecture-altering operations described in Section (19.3.4.3) to dynamically create, duplicate, and delete subroutines and loops during the run of GP. In addition, version 2.0 use the architecture-altering operations to dynamically create, duplicate, and delete recursions and internal storage. Since the architecture of the evolving program is automatically determined during the run, GPPS eliminates the need for the user to specify in advance whether to employ subroutines, loops, recursions and internal storage in solving a given problem. It similarly eliminates the need for the user to specify the number of arguments possessed by each subroutine. At last, it eliminates the need for the user to specify the hierarchical arrangement of the invocations of the subroutines, loops, and recursions.

19.3.4.5 Constraining syntactic structures

We saw in Section (19.3.2) that GP systems require type consistency where all sub-trees return data of the same type, ensuring that the output of any sub-tree can be used as one of the inputs to any other node. It ensures that the shuffling caused by sub-tree crossover, mutation, does not lead to incompatible connections between nodes. When problem domains have multiple types and do not naturally satisfy the type consistency requirement, we can consider creative

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

definitions of functions and implicit type conversion. However, it may not always be desirable. This is because GP systems that are constrained structurally or via a type system often generate results that are easier for humans to understand and analyse.
We discuss three different approaches to constraining the syntax of the evolved expression trees in GP:

1. simple structure enforcement,
2. strongly typed GP, and
3. grammar based constraints.

19.3.4.5.1 Enforcing particular structures If a particular structure is believed or known to be important then one can modify the GP system to require that all individuals to have that structure (see Koza [1992]). For example, in the periodic function $a\cos(bt)$, the GP is restricted to evolving expressions having that structure. That is, a and b are evolved freely, but the rest of the structure is fixed. Syntax restriction can also be used to make GP follow sensible engineering practices. For example, we might want to ensure that loop control variables appear in the correct parts of FOR loops and nowhere else. When implementing, we can ensure that all the initial individuals have that structure (for example, generating random sub-trees for a and b while fixing the rest), and then constrain operations like crossover so that they do not alter any of the fixed regions. Alternatively, we can evolve the various (sub)components separately in any of several ways. For example, evolve pairs of trees (a, b), or we can have two separate populations, one of which is being used to evolve candidates for a while the other is evolving candidates for b.

19.3.4.5.2 Strongly typed GP Since constraints are often driven by or expressed using a type system, a natural approach is to incorporate types and their constraints into the GP system . We want the system be closed so that all terminals are of one type and all functions can operate on this type and always yield a result also of this type.
In strongly typed GP (STGP) (see Montana [1995]), the system can contain multiple different types simultaneously. Every terminal has a type, and every function has types for each of its arguments and a type for its return value. The process generating the initial, random expressions, and all the genetic operators are then constrained not to violate the type system's constraints. Tight type checking can considerably reduce the size of the search space, reducing the effort to finding solution. This basic approach to types can be extended to more complex type systems including simple generics, multi-level type systems, and fully polymorphic, higher-order type systems with generics.

19.3.4.5.3 Grammar based constraints Another natural way of expressing constraints is via grammars, and these have been used in GP in a variety of ways (see Whigham [1996]). It use a grammar as a means of expressing the types of constraints discussed above. For example, we can enforce the structure for the periodic function using a grammar such as the following:

$$\begin{aligned}
\text{tree} &= E \times \cos(E \times t) \\
E &= \text{var} \mid E \text{ op } E \\
\text{op} &= + \mid - \mid \times \mid \\
\text{var} &= x \mid y \mid z
\end{aligned}$$

Genetic operators are restricted to only swapping sub-trees deriving from a common non-terminal symbol in the grammar. For instance, an E could be replaced by another E, but it could not be replaced by an op . It can be extended to context-sensitive grammars by incorporating various related concepts from computational linguistics (see Hoai et al. [2003]).
Another major area is grammatical evolution (see Ryan et al. [1998], O'Neill et al. [2003]). A grammar is used as in the example above, but instead of representing individuals directly using either expression or derivation trees, GE represents individuals using a variable length sequence of integers. For each production rule, the set of options on the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

right hand side are numbered from 0 upwards. In the example above the first rule only has one option on the right hand side; this would both be numbered 0. The second rule has two options, which would be numbered 0 and 1, the third rule has four options which would be numbered 0 to 3, and the fourth rule has three options numbered 0 to 2. An expression tree is then generated by using the values in the individual to choose which option to take in the production rules, rewriting the left-most non-terminal is the current expression.
In general, extra genetic material is simply ignored. Alternatively, sometimes a sequence can be too short in the sense that the end of the sequence is reached before the translation process is complete. There are a variety of options in this case, including failure (assigning this individual the worst possible fitness) and wrapping (continuing the translation process, moving back to the front of the numeric sequence).

19.3.4.5.4 Some limitations Limiting the search space by restricting the types of structures that can be constructed comes at a cost. It requires a complex machinery to support it and it makes it more difficult to generate type-correct individuals in the initial population as well as to find genetic operations that do not violate the type system. In general, one has to develop additional machinery in order to efficiently generate an initial population that satisfies the necessary constraints (see Montana [1995], Ratle et al. [2000]). Further, constraining the search space with the type system does not guarantee that it will make the evolutionary search process easier. For instance, it may not increase the density of solutions or (perhaps more importantly) approximate solutions. It might also make the search landscape more rugged by preventing genetic operations from creating intermediate forms on potentially valuable evolutionary paths. One might choose to extend solution density studies to the landscapes generated by typed systems in order to better understand the impact of these constraints on the underlying search spaces (see Langdon et al. [2002]). In the worst case, a constraint system could generate a needle-in-the-haystack situation, where the search space is indeed much smaller, but there is no smooth path to the solution. Thus, we might consider a transformation that actually increases the size of the search space, but also generates a gentle slope to the a solution. Hence, while types and other constraint systems can be powerful tools, we should explore the biases introduced by the constraints and not simply assume that they are beneficial to the search process.

19.3.4.6 Fitness evaluation

There are a number of issues associated with the fitness function ranging from the way that they are defined to how they are evaluated.

19.3.4.6.1 Absolute and relative fitness In general, the fitness of an individual is an absolute measure determined by running the individual against a set of test cases of fixed difficulty. In co-evolution, the test cases themselves are evolving programs which may be members of the same population or of different populations. Since the test cases are evolving too, the difficulty of the test cases would be changing from generation to generation (hopefully getting progressively more difficult). As a result, the fitness value of both the test program and the tested program are mutually dependent and hence relative as opposed to absolute. In turns, the knowledge of the optimal strategy no-longer required. Angeline et al. [1993] found that co-evolved players in a game of Tic Tac Toe developed better strategies than players using absolute fitness measures. Reynolds [1994] considered competitive fitness for the evolution of vehicle steering control programs used for the game of tag and reported near optimal solutions were found solely from direct competition between individuals in the population.

19.3.4.6.2 Multi-phase fitness Andre [1994] used GP to evolve programs that are capable of storing a representation of their environment (map-making), and then using that representation to control the actions of a simulated robot. He used GP to effectively co-evolve two programs (each making use of ADFs) for each individual. He split the fitness evaluation of each individual into two phases. In the first phase, the first program is run to extract and store features from the environment. Only the first program is provided with the set of functions to probe the environment. In the second phase, the second program is executed to use the stored representation to control the actions of a robot. The fitness value was awarded only for the success of this second phase. He was able to successfully evolve programs that

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

solve simple problems by examining their environment, storing a representation of it, and then using the representation to control action.

19.3.4.6.3 Full and partial fitness Badly behaved programs can be produced where there are recursive or iterative features, as these promote very long or even infinite loops. The presence of only one such program in the population halts the whole genetic programming system. In the case where a program may be expected not to halt (or take a very long time), Koza [1992] implemented ad-hoc loop limits which time out badly behaving individuals and give them poor fitness scores.
Maxwell [1994] proposed a more general mechanism to ensure programs which loop infinitely do not hang up the system. It applies an external time limit after which programs are interrupted if they are still running, thus allowing other members of the population to be run. The interrupted program is not aborted but is given a partial fitness which is used when selecting which programs to breed from and which are to be replaced by the new programs created. Programs which remain in the population are allowed to run for another time interval. Looping programs are given low partial fitness and so are eventually removed from the population, as new programs are bred.

19.3.4.6.4 Evaluation efficiency Individuals in a GP population are usually represented as a forest of trees. Handley [1994] represented the entire population of parse trees as a single directed acyclic graph (DAG). It brings immediate benefits in term of memory requirements since identical sub-trees do not need to be duplicated. In addition if the value computed for each sub-tree is cached, it does not need to be re-evaluated for each each instantiation, even if it occurs in a different generation. The important restrictions in this approach are that all functions must have no side effects and the fitness cases must be bounded and fixed for all individuals and all generations. With this approach Handley reported a 15-28 fold reduction in the number of nodes per generation and an 11-20 fold reduction in the number of nodes evaluated per run.

19.3.4.6.5 Pareto scoring Most genetic programming work uses a single fitness criterion namely the functionality of the program, ignoring issues such as size and run time. There has been some interest in using Pareto scoring (see Goldberg [1989], Langdon [1995]) which allows members of the population to be scored or compared using multiple criterion, such as functionality and runtime. Scoring systems which combine several criteria (such as functionality and efficiency) to yield a single fitness score have been tried with some success.

19.3.4.7 Other genetic operators

A number of alternatives to the standard GP crossover and mutation genetic operators have been proposed (see Kinnear [1994]). Some of them are:

- Hoist: The hoist operator creates a new individual entirely from a randomly chosen sub-tree of an existing individual. The operator is useful for promoting parsimony.
- Create: The create operator is unique in that it does not require any existing individuals. It generates an entirely new individual in the same way that an individual in the initial population is generated. This operator is similar to Hoist in that helps to reduce the size of program trees.
- Self-Crossover: It uses a single individual to represent both parents. The single individual can be selected using the standard fitness proportional selection or tournament selection methods.
- Modular/Cassette-Crossover: One of the restrictions of the standard crossover operator is that is not possible to swap blocks occurring in the middle of a tree path. It only allows entire sub-trees to be swapped. One of the reasons for the success of ADFs might be due to an effective relaxation of this restriction. This possibility lead Kinnear and Altenberg to develop the Modular or Cassette crossover operator. It can be seen as a module swap between two individuals. A module is defined in the program tree of the first individual, and another in the program tree of the second individual. The module in the first individual is then replaced by the one in the second

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

individual. The new module is then expanded. Clearly there is problem created by the possible mismatch of the arguments passed to the module (actual parameters) and the formal parameters defined by the module. There are two such possibilities, either the number of formal parameters are greater than the number of actual parameters or vice-versa. The former is resolved by extending the existing actual parameters with random chosen copies of themselves. The latter is resolved by choosing at random a subset of the existing actual parameters. Kinnear reports performance gains when any form of modular crossover is used with the Basic (non-ADF) GP approach, even with modular self-crossover.

- Constant perturbation: Spencer [1994] introduced the Constant Perturbation (CP) operator. It only affects special terminals called ephemeral random constants which are used to introduce constants into a GP individual (see Koza [1992]). The constant perturbation operator as used by Handley multiplies all constants in a GP individual by a random number between 0.9 and 1.1 (i.e. perturbs the constants by up to 10% of their original value). Handley reported two positive effects of the operator for the experiments that he performed. Firstly it allowed fine-tuning of the coefficients in programs which is otherwise difficult to achieve. Without CP, the only way of modifying the constant is to replace it via mutation or crossover, which tends have a more coarse effect. Secondly, constant perturbation enriches the set of terminals by adding new constants. Handley believes this operator to be an improvement over the ephemeral random constants constants used by Koza. Some of the experiments that he performed with symbolic regression showed dramatic improvements.

19.3.4.8 Developmental GP

When appropriate terminals, functions and/or interpreters are defined, standard GP can go beyond the production of computer programs. For instance, in cellular encoding (CE), programs are interpreted as sequences of instructions which modify (grow) a simple initial structure (embryo). Once the program terminates, the quality of the resulting structure is taken to be the fitness of the program. The primitives of the language must be appropriate to grow structures in the domain of interest. Typical instructions involve the insertion and/or sizing of components, topological modifications of the structure. It can be used to evolve neural networks (see Gruau [1994]), electronic circuits, and numerous other domains.
One of the advantages of indirect representations such as cellular encoding is that the standard GP operators can be used to manipulate structures (such as circuits) which may have nothing in common with standard GP trees. However, they require an additional genotype-to-phenotype decoding step, but when the fitness function involves complex calculations with many fitness cases the relative cost of the decoding step is often small.

19.3.4.9 Read's code

In GP, we code a parse tree t with Read's code: rooted trees composed of N vertices are coded by sequences of N non-negative integers

$$Code(t) = \left(\tau_1, ...\tau_N\right) \in \{0, 1, 2, ...\}^N$$

As an example, the interpretation of $code(t) = (201200)$ is given in Figure (19.20).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 19.20: Read's code

Theorem 19.3.1 *Necessary and sufficient conditions that a sequence of non-negative integers*

$$\left(\tau_1, ...\tau_N\right) \in \{0, 1, 2, ...\}^N$$

corresponds to a rooted tree are

$$\sum_{i=1}^{j} \tau_i \geqslant j \ , \ j = 1, 2, ..., N-1$$
$$\sum_{i=1}^{N} \tau_i = N - 1$$

Some advantages of Read's code in GP are:

1. Graph-theoretical structures: rooted trees are substituted by simple algebraic structure.
2. Simple algebraic manipulations with Read's code. For instance, looking for sub-trees is equivalent to looking for sub-codes.
3. Simple implementation in procedural programming languages (C++, Pascal).

19.3.4.10 Some improvements

GP can be hybridised with other techniques. For instance, Wong [2005] incorporated computer language grammars into GP. One can use simulated annealing and hill climbing to locally fine tune parts of solutions found by GP.
Tree GP can be made faster in a number of ways. For instance, we can reduce the number of times a tree is evaluated. Many applications find the fitness of trees by running them on multiple training examples. However, ultimately the point of fitness evaluation is to make a binary decision: does this individual get a child or not. In general, one consider a noisy selection technique. A large proportion of GP effort goes into adjusting the probability of the binary decision as to whether each individual in the population should be allowed to reproduce or not. If a program has already demonstrated it works very badly compared to the rest of the population on a fraction of the available training data, it is likely not to have children. Conversely, if it has already exceeded many programs in the population after being tested on only a fraction of the training set, it is likely to have a child. In either case, it is apparent that we do not need to run it on the remaining training examples (see Teller et al, [1997]).
In general, using all the training data all the time gives rise to a static fitness function. Arguably this tends to evolve the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

population into a cul-de-sac where the population is dominated by offspring of a single initial program. Considering high selection pressure decrease the time for the best individual to dominate the whole population. It is called the take over time (see Goldberg [1989]). This idea has been formalised (see Blickle [1996], Eiben et al. [2003]).
Gathercole et al. [1994] [1997] investigated a number of ways of changing which training examples to use as the GP progressed. There are several interacting effects:

1. by using only a subset of the available data, the GP fitness evaluation takes less time.
2. by changing which examples are being used, the evolving population sees more of the training data and, so, is less liable to over fit a fraction of it.
3. by randomly changing the fitness function, it becomes more difficult for evolution to produce an over specialised individual which takes over the population at the expense of solutions which are viable on other parts of the training data.

The most successful algorithm is the Dynamic Sub Selection (DSS) which has been incorporated into Discipulus.
In GP systems allowing for recursion or containing iterative elements, it is common to enforce a time limit, a limit on the number of instructions executed, or a bound on the number of times a loop is executed. Teller [1994] proposed the any time approach, where everyone in the population is run for the same amount of time. When the allotted time elapses the program is aborted and an answer extracted from it, regardless of weather it was ready or not. It suits graph or linear GP where it is easy to designate a register as the output register. The answer can be extracted from this register or from an indexed memory cell at any point (including whilst the programming is running).
A simple way to speed up the evaluation of complex fitness functions is to organise the latter into stages of progressively increasing computational cost. Individuals are evaluated stage by stage. Each stage contributes to the overall fitness of a program. However, individuals need to reach a minimum fitness value in each stage in order for them to be allowed to progress to the next stage and acquire further fitness. Often different stages represent different requirements and constraints imposed on solution.
Poli et al. [2005] [2006] proposed the backward chaining GP to reduce the number of fitness evaluations in runs of GP (and other EAs) using tournament selection with small tournament sizes. Tournament selection randomly draws programs from the population to construct tournaments, the winners of which are then selected. Although this process is repeated many times in each generation, when the tournaments are small there is a significant probability that an individual in the current generation is never chosen to become a member of any tournament. This is exploited not only to avoid the calculation of individuals that are never sampled, but also to achieve higher fitness sooner.

19.3.4.11 Fitness landscapes

We saw in Section (19.1.3.5) that the fitness landscape provides a powerful tool for visualising the difficulty of a problem. However, if we try using it in GP, we run into the problem of deciding which points are adjacent to each other. While we use the hamming distance in the fixed-length standard GA, this is not possible in the GP representation which is neither bit oriented nor is it of fixed length. The individuals in GP are program trees which must be compared in some way to determine how far apart they are to be plotted on the fitness landscape. Thus, this is a difficult problem.
Kinnear [1994b] examined the structure of the fitness landscape on which GP operates, and analysed the landscapes of a range of problems of known difficulty in order to determine the correlation between landscape measures and problem difficulty. The landscape analysis techniques that he employed include adaptive walks, and the autocorrelation of the fitness values of random walks. He found that the former showed better correlation with the problem difficulty than the latter for GP.

19.3.5 Other types of GP

We briefly present other types of GP where programs are represented in different ways than trees evaluated with an appropriate interpreter. Some example are linear programs and graph-like (parallel) programs.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

19.3.5.1 Linear GP

Since most architectures represent computer programs in a linear fashion (albeit with control structures, jumps and loops), we could directly evolve linear programs. Further, computers do not naturally run tree-shaped programs where slow interpreters have to be used as part of tree-based GP. On the contrary, by evolving the binary bit patterns actually obeyed by the computer, the use of an expensive interpreter (or compiler) is avoided and GP can run several orders of magnitude faster (see Nordin et al. [1999]).
In general, the crossover and mutation operators for linear GP ignore the details of the machine code of the computer being used. That is, crossover chooses randomly two crossover points in each parent and swaps the code lying between them (see Section (19.3.2)). Since the crossed over fragments are of different lengths, such a crossover may change the lengths of the programs (see Figure (19.21)). Further, computer machine code is organised into 32- or 64-bit words, so that the crossover points occur only at the boundaries between words. Thus, a whole number of words, containing a whole number of instructions are typically swapped over. Similarly, mutation operations normally respect word boundaries and generate legal machine code.

Figure 19.21: GP linear crossover: parents are the top two genomes

However, linear GP lends itself to a variety of other genetic operations such as homologous crossover and many other crossover and mutation operations (see Langdon et al. [2005]). See details in Figure (19.22).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 19.22: GP homologous crossover: top two parents crossover yield to child programs

When execution speed is essential, then the evolved code should be machine code for a real computer rather than some higher level language or virtual-machine code. Nordin [1994] started by evolving machine code for SUN computers. The Sun SPARC has a simple 32-bit RISC architecture which eases designing genetic operation which manipulate its machine code. Later, Nordin [1997] wrapped each machine code GP individual inside a C function. Each of the GP program's inputs were copied from one of the C function's arguments into one of the machine registers. There are only a small number of inputs. A small number (such as 2 to 4) of other registers are used for scratch memory to store partial results of intermediate calculations. Finally, the GP simply leaves its answer in one of the registers. The external framework uses this as the C function's return value. Then, Nordin ported his Sun RISC linear GP onto Intel's CISC. His work led to Discipulus (see Foster [2001]), which has been used from applications ranging from Bioinformatics to robotics and bomb disposal.
Note, linear programs can be interpreted, just as trees can be, since a linear interpreter can be readily implemented. A simple linear structure lends itself to rapid analysis, which can be used for dead code removal (see Brameier et al. [2001]). In some ways the search space of linear GP is easier to analyse than that of trees.

19.3.5.2 Graph based GP

Since trees are special types of graphs (see Section (13.1.1.2)), it is natural to extend GP so as to be able to evolve graph-like programs. Starting from the mid 1990s researchers have proposed several extensions of GP that do just that.
Poli [1996] proposed Parallel Distributed GP (PDGP) as a form of GP which is suitable for the evolution of efficient highly parallel programs which effectively reuse partial results. Programs are represented as graphs with nodes representing functions and terminals. Edges represent the flow of control and results. In its simplest form, edges are directed and unlabelled, in which case PDGP can be considered a generalisation of standard GP. However, more complex representations can be used, which allow for the exploration of a large space of possible programs including standard tree-like programs, logic networks, neural networks, recurrent transition networks and finite state automata. Programs are manipulated by special crossover and mutation operators which guarantee the syntactic correctness of the offspring, making the search is very efficient. PDGP programs can be executed in different ways, depending on whether nodes with side effects are used or not.
In Cartesian GP (see Miller [1999], [2006]), programs are represented by linear chromosomes containing integers. These are divided into groups of three or four, where each group is associated to a position in a 2D array. An element of

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

the group prescribes which primitive is stored at that location in the array, while the remaining elements indicate from which other locations the inputs for that primitive should be read. Hence, the chromosome represents a graph-like program, which is very similar to PDGP. The main difference being that Cartesian GP operators (mainly mutation) act at the level of the linear chromosome, while in PDGP they act directly on the graph.
One can also use non-graph-based GP to evolve parallel programs. Bennett [1996] used a parallel virtual machine in which several standard tree-like programs (called agents) would have their nodes executed in parallel with a two stage mechanism simulating parallelism of sensing actions and simple conflict resolution (prioritisation) for actions with side effects. Andre [1996] used GP to discover rules for cellular automata, a highly parallel computational architecture, which could solve large majority-classification problems. In conjunction with an interpreter implementing a parallel virtual machine, GP can also be used to translate sequential programs into parallel ones (see Walsh et al. [1996]) or to develop parallel programs.

19.3.6 Some applications

GP has been successfully used as an automatic programming tool, a machine learner or an automatic problem-solving machine and many more applications (see Figure (19.23)). We are going to present a few of these applications (see Poli et al. [2007] for more details).

Figure 19.23: GP applications

19.3.6.1 Symbolic regression

As discussed in Section (19.3.3), when solving a problem we need to define a proper fitness function. When only the side effects of the program are of interest, the fitness function usually compares the effects of the execution of a program in some suitable environments with a desired behaviour in an application-dependent manner. However, in some problems the goal is to find a function whose output has some desired property, such as matching some target values, or being optimum against some other criteria. This type of problem is generally known as a symbolic regression problem (SRP).
Symbolic regression, instead, consists in finding a general function (with its coefficients) that fits the given data points. Since GP does not assume a priori a particular structure for the resulting function, it is well suited to this sort of discovery task (see Koza [1992], Lew [2006], Cai et al. [2006]).
In addition to the five steps described in Section (19.3.3) In realistic problems, the collection of an appropriate set of data points is an important and sometimes complex task. For example, each point represents the (measured) values taken by some variables at a certain time in some dynamic process or in a certain repetition of an experiment. In case

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

of a evolving a soft sensor (see Jordaan et al. [2004]), we want to evolve a function that will provide a reasonable estimate of what a sensor (in a production facility) would report, based on data from other actual sensors in the system. This is typically done in cases where placing an actual sensor in that location would be difficult or expensive for some reason. Nonetheless, we need to place at least one instance of such a sensor in a working system in order to collect the data needed to train the model. Such experimental data typically come in large tables where numerous quantities are reported. Some of which are dependent variables (what we want to predict) while others are independent variables (used to make prediction). When the data included hundreds or even thousands of variables, the efficiency and effectiveness of any machine learning or program induction method, including GP, can dramatically drop as most of the variables are typically redundant or irrelevant. It is then necessary to perform some form of feature selection, where we decide which independent variables to keep and which to leave out.
There are other problems where more than one output (prediction) is required. In such problems we can use GP individuals including multiple trees, graph-based GP with multiple output nodes, linear GP with multiple output registers, a single GP tree with primitives operating on vectors.
When a suitable dataset is available, its dependent variables must all be represented in the primitive set. What other terminals and functions this will include depends on the type of the data being processed (numeric, strings, other) and is often guided by information available to the experimenter on the process that generated the data. If something is known (or strongly suspected) about the desired structure of the evolved function (for example, the data is known to be periodic, so the function should probably be based on a something like sin), then applying some sort of constraint may be beneficial.
In principle, the fitness function must measure the ability of each program to predict the value of the dependent variable given the values of the independent ones (for each data-point). Thus, most symbolic regression fitness functions tend to include sums over the (usually absolute or squared) errors measured for each record in the dataset.

19.3.6.2 Image and signal processing

Hampo et al. [1992] used GP for signal processing by evolving algorithms for preprocessing electronic motor vehicle signals for possible use in engine monitoring and control. Several applications of GP for image processing have been for military uses. Optical satellite images can also be used for environmental studies and for prospecting for valuable minerals. Smart et al. [2004] used local search (simulated annealing or gradient descent) to adjust or fine-tune constant values within the structure created by genetic search. Zhang evolved programs with GP to visually classify objects (typically coins). He has also applied GP to human speech. Parisian GP is a system in which the image processing task is split across a swarm of evolving agents (flies). The flies reconstruct three-dimensions from pairs of stereo images. As they buzz around in three-dimensions their position is projected onto the left and right of a pair of stereo images. The fitness function tries to minimise the discrepancy between the two images, thus encouraging the flies to settle on visible surfaces in the 3D space. Thus, the true 3D space is inferred from pairs of 2D image taken from slightly different positions.
While indexing the written word is now effective, it is not so simple when it comes to indexing speech and pictures. Theiler et al. [1999] worked at automatically indexing images. Extracting text from images (OCR) is almost a solved problem. With well formed letters and digits this is now done with near 100% accuracy as a matter of routine. A promising area is medical imaging. GP image techniques can also be used with sonar signals. Offline work on images, includes security and verification.

19.3.6.3 Financial trading

Edmonds et al. [1995] applied genetic programming of fuzzy logic production rules to financial trading. They implemented parse trees containing fuzzy operators (and, or, not) and fuzzy sets with linguistic values including greater than, less than, and other much larger than. Given historical data (open, high, low, close), they used the model to predict daily trading signals (long, short). Chen has extensively used GP in finance and economics. Recent papers include modelling of agents in stock markets (see Chen et al. [2005]), game theory, evolving trading rules for the S&P 500 (see Yu et al. [2004]), and forecasting the Heng-Sheng index (see Chen et al. [1999]).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Recall, the efficient markets hypothesis (EMH) is founded on the idea that everyone in a market has perfect information and acts rationally. However, without price differentials, there would be no money to be made from the market itself. Game theory has been a standard tool used by economists to try to understand markets but is increasingly supplemented by simulations with both human and computerised agents. GP is increasingly being used as part of these simulations of social systems. GP and computer tools have been used in a novel data-driven approach to try and resolve issues which were previously a matter of dogma. For example, Neely et al. [2001] used GP to study intraday technical trading of foreign exchange to suggest the market is efficient and found no evidence of excess returns. Kaboudan showed that GP could forecast international currency exchange rates (see Kaboudan [2005]), stocks (see Kaboudan [2000]), and stock returns (see Kaboudan [1999]). Dempster et al. [2000] [2001] at HSBC also used GP in foreign exchange trading. In 2007 Tony Brabazon and Michael O'Neill established the European workshop on evolutionary computation in finance and economics (EvoFIN). EvoFIN is held with EuroGP.

19.3.7 General comments

19.3.7.1 Fitness with caches

In computer hardware, data caches automatically hold copies of data locally in order to avoid the delays associated with fetching it from disk or over a network every time it is needed. It is generally used when a small amount of data is needed many times over a short interval. It can also be used to store results of calculations, avoiding the re-calculation of data. We have seen that GP populations have large amounts of common code generated by promoting fit individuals. We assume that each sub-tree has no side-effects, that is, its results pass through its root node in a well organised and easy to understand fashion. We can therefore store the input (root node) and output of a sub-tree, avoiding re-executing the code whenever we are required to re-run the sub-tree in crossover (different individual) or mutation (different time). In GP, one must store this information and let the interpreter know that the inputs of the sub-tree are the same. Assuming that every tree in the population is run on exactly the same inputs for each of the fitness cases, only the fixed set of test cases matter. Thus, we can store a vector of values returned by each sub-tree with length equals to the number of test cases. When sub-trees are crossed over the cache of the sub-tree remains valid and so cache values can be crossover like the code.
When some code is inserted into an existing tree via mutation or crossover, the new code will most likely be different from the old code so that the caches of every node between the new code and the root node may be invalid. The simplest thing is to re-evaluate them all. Note, the caches in all the other parts of the individual remain valid and so can be used when the cache above them is re-evaluated. Thus, if the crossed over code is inserted at level d only d nodes need to be evaluated. Note, we use hashing schemes to efficiently find common code.
Handley [1994] proposed the DAG implementation incorporating fitness caches. Keijzer [1996] presented an efficientl way of representing populations in genetic programming. McPhee et al. [1998] proposed Sutherland as a GP system which has implemented fitness caches. Langdon [1998] used fitness caches in evolved trees with side effects by exploiting syntax rules about where in the code the side-effects could lie.

19.3.7.2 Some recommendations

We present some of the recommendations proposed by Koza [1992], Poli et al. [2007].

- GP populations should be closely monitored as they evolve. Some properties can be easily measured, giving indications about potential problems:
  - Frequency of primitives: Recognising the loss of a primitive from the population (or its frequency has fallen to a low level, consistent with the mutation rate) may help to diagnose problems.
  - Population variety: If the variety (the number of distinct individuals in the population) falls below 90% of the population size, it indicates there may be a problem. However, a high variety does not mean the reverse. GP populations often contain introns, and so programs which are not identical may behave identically.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Being different, these individuals contribute to a high variety. Measuring phenotypic variation (diversity of behaviour) may also be useful.

- Measures should be taken to encourage population diversity. Panmictic steady-state populations with tournament selection, reproduction and crossover may converge too readily. The above-mentioned metrics may indicate if this is happening in a particular case. Some solutions include:
  - Not using the reproduction operator.
  - Addition of one or more mutation operators.
  - Smaller tournament sizes and/or using uniform random selection (instead of the standard negative tournaments) to decide which individuals to remove from the population. However, the latter means the selection scheme is no longer elitist. In some cases, it may be worthwhile forcing it to be elitist.
  - Splitting large populations into semi-isolated demes.
  - Using fitness sharing to encourage the formation of many fitness niches (see Section (19.1.3.6)).
- Use of fitness caches (either when executing an individual or between ancestors and children) can reduce run time and may repay the additional work involved with using them (see Section (19.3.7.1)).
- Where GP run time is long, we should periodically save the current state of the run. Should the system crash, the run can be restarted from part way through rather than at the start. Care should be taken to save the entire state, so restarting a run does not introduce any unknown variation. The bulk of the state to be saved is the current population via compression(such as gzip).

We saw that GP was about creating visible programs, so that when presenting results we should display the whole evolved program. Note, once we have found a model we want to make sure it is small and comprehensible. One solution is to continue evolution changing the fitness function to include a second objective: that the model be as small as possible. GP can then trim the trees but ensure the evolved program still fits the training data.

19.4 Multi-objective optimisation problems

Multi-objective Optimisation Problems (MOPs) are those problems where several conflicting objective functions must be optimised simultaneously. MOPs with more than three objective functions are usually known as Many-objective Optimisation Problems (MaOPs) in the related literature. The solution to MOPs and, consequently, MaOPs is a set of trade-off points referred to as the Pareto optimal front or Pareto optimal set. In the general case however, finding the solution of a MOP is an NP-complete problem. As a result, heuristic or meta-heuristic methods, such as Multi-objective Evolutionary Algorithms (MOEAs), have arisen as one of the most popular techniques to successfully address MOPs.
A Multi-objective Optimisation Problem (MOP) can be defined as the problem in which a set of objective functions $f_1(x), ..., f_M(x)$ should be jointly optimised

$$\min F(x) =< f_1(x), ..., f_M(x) > \ , x \in \mathcal{S}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$ are (possibly) conflicting objective functions, and $\mathcal{S} \subseteq \mathbb{R}^n$ is known as the feasible set and can be expressed as a set of restrictions over the decision set, in our case, $\mathbb{R}^n$. For consistency, we have transformed all the maximisation problems of the type $\max f_i$ into equivalent minimisation problems $\min(-f_i)$. The image set of $\mathcal{S}$ S produced by function vector $F(\cdot)$ is called the feasible objective set or criterion set.
In most cases there are infinitely many optimal solutions, so that an optimal solution is a solution where there exists no other feasible solution that improves the value of at least one objective function without deteriorating any other objective. This is the notion of Pareto optimality. That is, the solution to these types of problems is a set of trade-off points. The optimality of a given solution can be defined in terms of the Pareto dominance relation. We restrict our

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

attention mainly to the case of convex objectives and convex feasible region.
From a mathematical point of view, every Pareto optimal solution is equally accept-able as the solution to the multi-objective optimisation problem. However, for practical reasons only one solution shall be chosen at the end. Picking a desirable point out of the set of Pareto optimal solutions involves a decision maker (DM) who is able to express preference relations between different solutions. In general MOPs should be solved in two steps executed in sequence:

1. An approximation of the Pareto set is determined, that contains as many non-dominated solutions as possible.
2. A solution is chosen among these Pareto-optimal solutions.

Sorensen [2010] showed that the sequential approach to MOPs was strongly outperformed by a simultaneous approach, in which the preferences of the decision maker were taken into account during the multi-objective optimisation.

Before considering Evolution Algorithms (AEs) for solving MOPs, one can try simpler approaches such as the weighted sum method and the hierarchical method. The weighted sum method allows the MOP to be cast as a single-objective mathematical optimisation problem. The problem can reformulated as

$$\min \sum_{i=1}^{M} w_i f_i(x) \text{ s.t. } x \in \mathcal{S}$$

where $w_i \geqslant 0$ and $\sum_{i=1}^{M} w_i = 1$. The weights of each objective function are assigned by the DM based on the intrinsic knowledge of the problem. There are several ways for normalising the weights. A slight rewriting of the weighted sum method is the hierarchical method which allows the decision maker to rank the objective functions in a descending order of importance, from 1 to M. Each objective function is then minimized individually subject to a set of additional constraints that do not allow the values of each of the higher ranked functions to exceed a prescribed fraction of their optimal values obtained on the corresponding steps. That is, once the optimal value for the specific objective has been obtained, the possible values this objective function can take on the following steps are restricted to a box (or a ball) around this optimal solution. Depending on the size of the restriction this strategy effectively puts more weight on the higher ranked objective functions. As an example, we consider the problem with two objective functions and assume that f_2 has higher rank than f_1. We then solve

$$\min f_2(x) \text{ s.t. } x \in \mathcal{S}$$

to find the optimal objective value f_2^*. Next we solve the problem

$$\begin{aligned} &\min f_1(x) \\ &\text{s.t. } x \in \mathcal{S} \\ &f_2(x) \leqslant f_2^*(x) + \epsilon \end{aligned}$$

Here, f_2 is more important than f_1 and we do not want to sacrifice more than $X\%$ of the optimal value of f_2 to improve f_1. The hierarchical method provides the decision maker with another way of specifying the relative importance of the objective functions.

19.4.0.1 Defining truly multi-objective problems

When the optimisation surface is poorly behaved, it is far more difficult to find the optimum using a sequential approach than it is for well behaved surface. For example, we denote Z the dimension up the Figure (19.24), and let it be the function to be optimised. If you start by optimising Y at any fixed value in X, and then fix Y and search for optimum X, or X at any fixed value of Y, and then search for the optimum Y, we will not find the maximum except by chance.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 19.24: MOP Optimum

Having several objectives is not enough for a problem to be truly multi-objective. Conflict among the objectives (at least some with respect to some others) is required. In the case where there is conflict among the objectives, it is very easy to come up with example where optimising sequentially is sub-optimal. This is observed by considering any extremes of the Pareto front. For example, if minimising objective X maximises objective Y, and they are in conflict. On one extreme we will have the minimum value for objective X with the maximum possible value for objective Y. In the other extreme, we will have the opposite, the minimum value for objective Y with the maximum value for objective X.

19.4.0.2 Using genetic algorithms

We saw in Section (19.3) that genetic algorithms (GAs) enabled optimum of highly nonlinear objective functions. Basic GAs use chromosomes that are integers or Booleans, so well suited for problems mixing real values with integer values. When considering constraints, they can be coded using some kind of rule engine, or hard coded if they are never likely to change. Further, Evolutionary Algorithms (EAs) are well known for solving multi-objective optimisation problems (MOPs). Coello Coello et al. [2001] proposed a multiobjective optimisation approach based on a micro genetic algorithm (micro-GA), which is a genetic algorithm with a very small population and a reinitialisation process. Selection methods are responsible for both convergence and diversity of the solution set provided, and as such are a key component of all multi-objective optimisation problem. Different types of MOEAs have been proposed for solving MaOPs as an alternative to Pareto-based MOEAs: decomposition-based MOEAs, grid-based MOEAs and indicator-based MOEAs. For instance, Gomez et al. [2013] considered the incorporation of performance indicators as the selection mechanism of a multi-objective evolutionary algorithm (MOEA). They used the $R2$ indicator, which presents some advantages with respect to the hypervolume, the main one being its low computational cost. When the MOP has a small number of objectives function, and when the problem is discrete, we can use the NSGA-II with binary encoding (see Kundu et al. [2015], Shao et al. [2016], Lezcano et al. [2020]). Marti et al. [2017] performed a comprehensive analysis of state-of-the-art selection method and found that the reference-point-based selection scheme of the Non-dominated Sorting Genetic Algorithm III (NSGA-III) and a modified version of the Non-dominated Sorting Genetic Algorithm II (NSGA-II), where the crowding distance is replaced by the Euclidean distance, yield the best results.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 20

Evolving networks

As discussed in this book, many ML and control problems can be cast as optimisation problems, where a general function approximator is optimised (see Chapter 10). Thus, it is logical to relate neural networks (NNs) and genetic algorithms (GAs), which demonstrate powerful problem solving ability. Neural networks are composed of interconnected nodes (neurons) that compute values based on external inputs. They operate by simulating a collection of neuron-like components sending signals to each other. To get an intelligent network, the challenge consist in deciding what should be the topology as well as the connection weights. Thus, the choice of the basic parameter (network topology, learning rate, initial weights) often already determines the success of the training process. This is the problem of learning (see Barto [1990]):

1. parametric learning: learning weight values.
2. structural learning: learning topology of nodes and links.

In practice, the selection of these parameter follow rules of thumb, but their value is at most arguable. We identify two categories:

1. Constructive algorithms: Assume a simple network and add nodes and links (see Ash [1989]).
2. Destructive algorithms: Start with a large network and prune off superfluous components (see Mozer et al. [1989], Cun et al. [1990]).

These algorithms monotonically modify the network structure, which limit the traversal of the available architectures since once the latter has been explored and determined to be insufficient, a new architecture is adopted and the previous one becomes topologically unreachable. Further, these methods only use a single predefined structural modification, such as add a fully connected hidden unit, to generate successive topologies. The answer for a natural brain is evolution over million of years. Genetic algorithms and similar theories are global search methods, that are based on principles like selection, crossover and mutation. The goal of neuroevolution is to use these algorithms to trigger a similar evolutionary process inside a computer. That is, it focuses on the origin of the architecture of the brain itself. This form of structural hill climbing can lead to solutions trapped in local minima. At last, these algorithms make simplifying architectural assumptions to facilitate the network induction, resulting in tasks being forced into the architecture. Preferably, the constraints for such solution should come from the task rather than be implicit in the algorithm.

Neuroevolution, or neuro-evolution, is a form of artificial intelligence that uses evolutionary algorithms to generate artificial neural networks (ANN), parameters, topology and rules. It is most commonly applied in artificial life, general game playing and evolutionary robotics. The main benefit is that it can be applied more widely than supervised learning algorithms, which require a syllabus of correct input-output pairs. In contrast, neuroevolution requires only a measure of a network's performance at a task. For example, the outcome of a game (whether one player won or lost) can be

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

easily measured without providing labelled examples of desired strategies. Neuroevolution can be contrasted with conventional deep learning techniques that use gradient descent on a neural network with a fixed topology.

The number and variations of neuroevolution algorithms proposed is very large. One common distinction is between these algorithms is as follows:

1. Fixed topology: Algorithms that evolve only the strength of the connection weights for a fixed network topology (sometimes called conventional neuroevolution). The genes of the evolving NNs encode their weights frozen from birth.

2. Algorithms that evolve both the topology of the network and its weights. Some of theses algorithms are called TWEANNs, for Topology and Weight Evolving Artificial Neural Network algorithms. Among these algorithms there is NEAT for Neuroevolution of Augmenting Topologies.

A separate distinction can be made between

1. The methods that evolve the structure of ANNs in parallel to its parameters (those applying standard evolutionary algorithms).

2. The methods that develop them separately (through memetic algorithms).

Note, the automatic designing of the deep neural network architecture using reinforcement learning, were recently attempted (see Baker et al. [2017]).

20.1 Introduction to neuroevolution

Neuroevolution is a sub-field within artificial intelligence (AI) and machine learning (ML) that consists of trying to trigger an evolutionary process similar to the one that produced our brains, except inside a computer. It seeks to develop the means of evolving neural networks through evolutionary algorithms.
Taking into consideration the anticipated impact on the fields of neuro-control, neuro-cybernetics and evolutionary robotics, the number and variations of neuroevolution algorithms proposed is very large. Each evolutionary method has its own benefits for a specific application area, where it outperforms other methods. For instance, in the field of robotic, evolving a network to control some movements (say walk) requires a simulator because neuroevolution takes a lot of trials. That is, the NN control a simulated movement of the robot. Using such a simulator led researchers to propose various models which we will briefly introduce.

20.1.1 The framework

20.1.1.1 Types of topology

Evolutionary approaches for artificial neural networks (ANNs) can be mainly divided into two types:

1. Fixed topology algorithms: They works with networks having a fixed number of fully interconnected neurons or with layers of neurons with a fixed interconnection pattern. However, the chosen number of neurons and their interconnection structure may not be appropriate and thus may not allow a solution at all. Further, the search space is usually very large right from the beginning, because all potential neurons and synapses of the network are always included in its maximal dimensions. Accordingly, it is difficult to find the optimal balance between a minimal search space and a topology that in the first place includes solutions to the problem. Examples of research focusing on the estimation of parameters for a fixed network architecture are Whitley et al. [1990], Beer et al. [1992].

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. Topology evolving algorithms: They have the ability to start with a small network (see Elman [1993]) and to add neurons and synapses to gradually increase the search space slowly over time or when it seems necessary (see Hulse et al. [2004], Stanley [2004]). Such algorithms keep the search space small without restricting the size and topology of the network over the course of the evolution. So, if solutions in small networks cannot be found, then the evolution can continue to search in larger network topologies, so that the initial network topology can be chosen much less thoroughly. Many topology evolving algorithms can also be used to shrink already well performing larger solutions to smaller, easier comprehensible and easier to generalise network structures (see Giles et al. [1994])).

20.1.1.2 Genome encoding

Evolutionary algorithms operate on a population of genotypes (also referred to as genomes). An important design choice in neuroevolution (NE) is the genetic representation (genotype) of the network that the combination and mutation manipulate. In NE, a genotype is mapped to a neural network phenotype that is evaluated on some task to derive its fitness. For example, an ANN with weights represented by four binary bits, where zero weight implies no connection, is presented in Figure (20.1). The genome representation of neural networks differs widely between the various algorithms. Common representations are matrices of synaptic weights, object structures of parametrised neuron types, lists of weights and nodes, binary strings, graph descriptions, trees and lists of construction rules, and much more.

Figure 20.1: ANN and its genotype representation

The genome encodings are commonly separated into two classes:

1. Direct encoding: The genotype directly maps to the phenotype. That is, every neuron and connection in the neural network is specified directly and explicitly in the genotype. They directly represent the weights of all synapses of the network. One advantage of this approach is its simplicity, which makes it easy to use and implement. Further, most directly encoded networks are free of a representation bias, because all kinds of networks can be encoded with this method. The mapping between genotype and phenotype is simple and works fine in both directions. This is relevant, if one wants to evolve a network structure not from scratch, but instead on top of an existing, manually prepared network structure. NEAT is an example of direct encoding.

2. Indirect encoding: The genotype specifies indirectly how the network should be generated. They are used to achieve several aims such as

   - modularity and other regularities.
   - compression of phenotype to a smaller genotype, providing a smaller search space.
   - mapping the search space (genome) to the problem domain.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The genotypes evolve parameters that are used to indirectly derive the network topology and/or synaptic weights. In that case, the DNA is a compressed representation of the brain. In general, it involves the application of grammars or rule sets to construct the network. Such algorithms are often referred to as

- Developmental algorithms (see Gruau [1995], Bongard et al. [2003], Downing [2007]), or
- Algorithms with artificial embryogeny (Bentley et al. [1999], Bongard et. al. [2001], Stanley et al. [2003]).

Developmental approaches have the advantage that the genome size is often much more compact compared to direct encoding if applied to larger networks. This can speed up evolution, but usually also induces a strong representation bias to the evolution, because often not all kinds of networks can be encoded (with similar likeliness) with this type of algorithm. Also, whereas the construction of a phenotype from a given genotype is relatively easy using a developmental algorithm, the definition of the genotype from a given phenotype is often difficult and complicated. In addition, even the construction of a phenotype from a given genotype can already be time-consuming.

Some indirectly encoding NE algorithms, like simulated growing synapses (see Nolfi et al. [1991]) or those using compositional pattern-producing networks (CPPNs) (see Stanley [2007]), may even slow down evolution, because the mapping from genotype to phenotype is computationally expensive.
Traditionally, indirect encodings that employ artificial embryogeny have been categorised along the lines of a grammatical approach versus a cell chemistry approach. The former evolves sets of rules in the form of grammatical rewrite systems. The latter attempts to mimic how physical structures emerge in biology through gene expression. Indirect encoding systems often use aspects of both approaches.
For instance, Stanley et al. [2003] proposed a taxonomy for embryogenic systems that is intended to reflect their underlying properties. The taxonomy identifies five continuous dimensions, along which any embryogenic system can be placed:

- Cell (neuron) fate: the final characteristics and role of the cell in the mature phenotype. This dimension counts the number of methods used for determining the fate of a cell.
- Targeting: the method by which connections are directed from source cells to target cells. This ranges from specific targeting (source and target are explicitly identified) to relative targeting (e.g. based on locations of cells relative to each other).
- Heterochrony: the timing and ordering of events during embryogeny. Counts the number of mechanisms for changing the timing of events.
- Canalisation: how tolerant the genome is to mutations (brittleness). Ranges from requiring precise genotypic instructions to a high tolerance of imprecise mutation.
- Complexification: the ability of the system (including evolutionary algorithm and genotype to phenotype mapping) to allow complexification of the genome (and hence phenotype) over time. Ranges from allowing only fixed-size genomes to allowing highly variable length genomes.

Indirect encoded ANNs can be used to evolve robot gaits since their regular connectivity patterns tend to support the regularity of motions involved in walking or running.

20.1.1.3 Crossover

We presented in Section (19.1.3.4) the dual representation in the genetic algorithm and its associated interpretation function. Problems arise with crossover when evolving neural networks in a deceptive environments (see Goldberg [1989b]). This is because the (arbitrary) combination of two well performing neural networks most often creates

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

networks that behave very differently from their parents. One reason is that the same functionality can be produced by very different neural structures, so that two equally well performing networks may be structurally very different. We distinguish three forms of deception when using crossover to evolve connectionist networks:

1. Networks sharing both a common topology and common weights: Two such networks need not have the same bit string representation. Crossover will then tend to create offspring that contain repeated components, and lose the computational ability of some of the parents' hidden units, resulting in less fit networks. See Figure (20.2). This is the competing conventions problem, where the number of competing conventions grows exponentially with the number of hidden units (see Belew et al. [1992], Hancock [1992], Schaffer et al. [1992]).

2. Two networks with identical topologies but different weights: A single connectionist topology affords multiple solutions for a given task, each implemented by a unique distributed representation spread across the hidden units. Since the removal of a small number of nodes has been shown to effect only minor alterations in the performance of a trained network, the computational role played by each node in the overall representation of the task solution is determined purely by the presence and strengths of its interconnections. Further, there is no need for correlation between distinct distributed representations over a particular network architecture for a given task. It reduces the chance that an arbitrary crossover operation between distinct distributed representations will construct viable offspring regardless of the interpretation function used.

3. The parents differ topologically: The types of distributed representations that can develop in a network vary widely with the number of hidden units and the network's connectivity. Thus, the distributed representations of topologically distinct networks have a greater chance of being incompatible parents. This further reduces the likelihood that crossover will produce good offspring.

As a result, the interpretation function must compensate for all the type of deceptiveness for crossover to be a viable operator. Thus, crossover is rarely used for the evolution of neural networks (NNs), and better results should be expected with reproduction heuristics that respect the uniqueness of the distributed representations.
While combining networks of type [1] usually destroys the functionality of both networks, so that the created network performs much worse than its parents, algorithms that use variants of this simple crossover include Angeline et al. [1994], Bongard [2003], Doncieux et al. [2003].
Another algorithm that introduced crossover to neuro-evolution is the NEAT algorithm (see Stanley et al. [2002a] [2002b]) and all the derivatives of that algorithm (for example HyperNEAT (Stanley et al. [2009]), Modular NEAT (see Reisinger et al. [2004]), NEON (see Inden [2007] [2008]), NEATfields (see Inden et al. [2010])). These algorithms allow disparate topologies to cross over in a meaningful way. The crossover successfully enhances the evolution performance, because it does not combine arbitrary network areas of two networks, but instead uses historical markings to find exchangeable sections with comparable evolutionary history by lining up genes with the same origin. Exchanging such network sections is much less destructive than arbitrary crossover. Stanley [2004] showed that using crossover in NEAT improve the evolution process measurably.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 20.2: Distinct bit strings A and B map to equivalent networks that assign the hidden units in different orders. Here, the offspring contains multiple copies of the same hidden node.

20.1.1.4 Basic algorithm

A basic algorithm is as follows (see Floreano et al. [2008]): A population of genotypes that encode NNs is evolved to find a network (weight and/or topology) that can solve a computational problem. Typically each genotype is encoded into a neural network, which is then tested on a specific task for a certain amount of time. The performance or fitness of this network is then recorded and once the fitness values for the genotypes in the current population are determined, a new population is generated by slightly changing the ANN-encoding genotypes (mutation) or by combining multiple genotypes (cross-over). In general, genotypes with a higher fitness have a higher chance of being selected for reproduction and their offspring replaces genotypes with lower fitness values, thereby forming a new generation. This generational loop is typically repeated hundreds or thousands of times, in the hopes to finding better and better performing networks.

20.1.1.5 Some algorithms

There is a large variety of neuroevolution (NE) algorithms (see Yao [1999]). While some evolve only the connection weight values of fixed topology networks (see Saravanan et al. [1995], Gomez et al. [1999]), others evolve both weights and network topology simultaneously (called TWEANNs). The neuroevolution methods, with direct or indirect encoding, also consider different evolutionary algorithms. Some examples are as follows:

- Neuro-genetic evolution (see Ronald et al. [1994]): It uses direct encoding with genetic algorithm.
- Cellular encoding (CE) (see Gruau [1994]): It uses indirect encoding with genetic programming (GP).
- GeNeralized acquisition of recurrent links (GNARL) (see Angeline et al. [1994]): It uses direct encoding with evolutionary programming (EP). It makes minimal architectural restrictions and avoid structural hill climbing by performing both structural and parametric learning.
- EPNet (see Yao et al. [1997]): It uses direct encoding with evolutionary programming combined with back-propagation and simulated annealing.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Neuroevolution of augmenting topologies (NEAT) (see Stanley [2004]). It uses direct encoding with genetic algorithm (GA). It combines the usual search for the weights with the evolution of increasingly complex neural networks (it complexifies the network structure). It tracks genes with historical markings allowing crossover between different topologies, and protects innovation via speciation.
- Hypercube-based neuroevolution of augmenting topologies (HyperNEAT) (see Stanley et al. [2009]): It uses indirect non-embryogenic. Spatial patterns generated by a compositional pattern-producing network (CPPN) within a hypercube are interpreted as connectivity patterns in a lower-dimensional space. This is done with genetic algorithm where the NEAT algorithm is used to evolve the CPPN.
- Evolvable substrate hypercube-based neuroevolution of augmenting topologies (ES-HyperNEAT) (see Risi et al. [2012]): It uses the same indirect encoding as in HyperNEAT. The same evolutionary algorithm is also used.
- Interactively constrained neuroevolution (ICONE) (Rempis [2012]): It uses direct encoding and includes constraint masks to restrict the search to specific topology and parameter manifolds. This is done with evolutionary algorithm (EA).
- Spectrum-diverse unified neuroevolution architecture (SUNA) (Vargas et al. [2017]): It uses direct encoding and consider the unified neural representation. It is done with genetic algorithm and uses a diversity preserving mechanism called spectrum-diversity that scales well with chromosome size, is problem independent and focus more on obtaining diversity of high level behaviours. The concept of chromosome spectrum is used together with a novelty map population (MPM).
- Modular agent-based evolver (MABE) (see Rostami et al. [2017]): It uses a direct or indirect encoding of Markov networks, neural networks and other arbitrarily customisable controllers. All sorts of evolutionary algorithms can be used. Covariance matrix adaptation with hypervolume sorted adaptive grid algorithm (CMA-HAGA): It uses direct encoding including an atavism feature which enables traits to disappear and re-appear at different generations. This is done with a multi-objective evolution strategy with preference articulation.

20.1.2 Improving the search

20.1.2.1 Diversity

Evolution benefits from the conservation of a certain level of diversity since networks with very different topologies and weight distributions can show a similar fitness rating. Furthermore, even small enhancements of a behaviour may require large changes to the network structure. Thus, evolution often has to first cross subspaces of the search space with a lower fitness to end up with a higher fitness. That is, it has to cross from one local optimum to another. This is because a premature convergence to a single class of solution candidates easily leads to an invincible local optimum. Some techniques to encourage diversity measures are increasing mutation rates when the rate of performance improvement stagnates (see Russell et al. [2009]), explicitly selecting for genetic diversity (see Stanley et al. [2002a]), or behavioural diversity (Mouret et al. [2012]), or changing the structure of the population (Whitley et al. [1999]). Although the diversity can be influenced at the selection level (see Horn [1997]) or with classical diversity measures such a fitness sharing, niching (see Sareni et al. [1998]), some neuro-evolution methods provide specific features to increase the diversity.
For instance, SANE (see Moriarty et al. [1994]) and ESP (see Gomez [2003]) increased the diversity by not evolving complete neural networks, but instead by evolving single neurons, that are assembled to a full network for each evaluation (cooperative co-evolution). That is, individual neurons of a network might be evolved in their own sub-populations to cooperate with other neurons and then grouped with neurons from other sub-populations to form full working networks. Since the neurons are always combined in other ways, the resulting networks remain diverse longer. In addition ESP makes use of fixed niches that further enhance diversity. Using a similar approach on the synaptic level Gomez et al. [2008] [2008] proposed CoSyNE which also uses niches to delay convergence.
Alternatively, to protect different lineages during evolution, Stanley [2004] proposed NEAT which uses a niching

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

method based on historical markings. It marks genes with a historical marking to ensure a coherent result of crossover (see Section (20.1.1.3)). It allows a dynamic grouping of the networks into similarity classes, such that only similar networks have to compete against each other. It has the capability of evolving increasingly deep networks.

20.1.2.2 Novelty and quality diversity

It is sometimes detrimental to the long term success of evolution to select the reproducing parents only from fitness criteria. The idea being that parents should not always be selected based on their objective performance, but rather on their novelty. For instance, when considering walking robot, it was argued that the more interesting concept (oscillating) was more likely to lead to robust walking than the currently best strategy (lurching). It turns out that oscillation is an essential stepping stone to robust walking, so that it can actually be better to ignore your objective and select simply for novelty rather than for improving objective performance. It led to the novelty search algorithm (see Risi et al. [2009], Lehman et al. [2008] [2011a]), which is a selection scheme based on the idea that you choose the most novel as parents instead of choosing the most fit.
Almost all the effort on selection over the history of evolutionary computation has focused on selecting parents based on their fitness. We should not only choose the most fit, because that would quickly drain diversity from the population, even if the more fit tend to have a higher chance of reproducing than the less fit. In novelty search, if the new candidate's behaviour is significantly different from what has come before, it has a high chance of being selected to reproduce, regardless of how fit it is.
It has led to a whole new research area called quality diversity (sometimes illumination algorithms). In biological terms, they illuminate the phenotype-fitness map (see Bull et al. [2011]). This new class of algorithms, generally derived from novelty search, aims not to find a single optimal solution but rather to illuminate a broad cross-section of all the high-quality variations of what is possible for a task, like all the gaits that can be effective for a quadruped robot (see Mouret [2011], Clune et al. [2013] (MOLE)). One such algorithm, the Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) proposed by Mouret and Clune, is about finding all the interesting possibilities that exist within the search space. Quality diversity algorithms (the next step beyond novelty search) add the notion of fitness back in, but in a careful way so as not to undermine the delicate push for novelty. The result is a class of algorithms that diverge through the search space, while still pulling out the best of what is possible.
It was on the cover of Nature, in an article by Cully et al. [2015], for the discovery of just such a large collection of robot gaits, which can be selectively called into action in the event the robot experiences damage. The MAP-Elites algorithm illuminates search spaces, allowing researchers to understand how interesting attributes of solutions combine to affect performance, either positively or, equally of interest, negatively (see Mouret et al. [2015]). It is inspired by two previous illumination algorithms, Novelty Search plus Local Competition $(NS + LC)$ (see Lehman et al. [2011b]) and the Multi-Objective Landscape Exploration algorithm (MOLE) (see Clune et al. [2013]).
The algorithm provides a holistic view of how high performing solutions are distributed throughout a search space. It creates a map of high-performing solutions at each point in a space defined by dimensions of variation that a user gets to choose, revealing relationships between dimensions of interest and performance. The search for a solution in any single cell is aided by the simultaneous search for solutions in other cells. This parallel search is beneficial because (1) it may be more likely to generate a solution for one cell by mutating a solution to a more distant cell, called goal switching (see Nguyen et al. [2015]), or (2) if it is more likely to produce a solution to a cell by crossing over two solutions from other cells.

20.1.2.3 Network shaping

Shaping (also called iterative or incremental evolution) is a common strategy (Bongard [2003], Miikkulainen et al. [2006], Mouret et al. [2008]) used to simplify the evolution of a complex task by subdividing it into multiple, easier to solve sequential experiments and to combine their results. It can be used with almost all neuro-evolution algorithms, because it only requires the iterative conduction of separate evolution experiments and the ability to evolve new structures on top of previously evolved ones.
Network shaping refers to the ability of a NE algorithm to allow shaping on the network level. This means that

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

the network itself, not only the experimental settings, can be influenced with external measures (based on domain knowledge) to evolve structures iteratively in certain orders and organisations. It is very useful if we want to guide evolution into promising regions of the search space.
Examples are HyperNEAT (see Stanley et al. [2009]) and HybrID (see Clune et al. [2009]), where the evolving structures can be influenced by different choices of the positions and distribution of (sensor and motor) neurons. The same holds for algorithms with simulated growing synapses (see Nolfi et al. [1991]). In these algorithms, an elaborated choice of the neuron positions can significantly influence the network topology and the evolution success. A simple simple approach is provided by the ENS3 algorithm (see Dieckmann [1995], Hulse et al. [2004]). It provides a protection mechanism that allows the prevention of all mutations from selected neurons, so that (initial) network structures can selectively be included or excluded from mutations over the course of a series of consecutive experiments. More control over the evolving network structures is provided by the SGOCE method (see Kodjabachian [1998], Meyer et al. [2003]). It is a developmental algorithm, where the user can choose the number of evolving structure construction programs and specify in an editor, at which positions of the network these programs start to construct their sub-networks. Also, the orientation of the initial seeds of these programs can be specified, which allows the definition of symmetric, modular networks, which is especially useful for repetitive structures, such as multi-legged walking machines.
A different approach uses the modular evolution algorithm ModNet (Doncieux et al. [2004a]), where the networks are composed of non-hierarchical evolvable neuro-modules with one or two fixed input and output neuron. Whereas the network structure of these modules can be fully evolved, it is also possible to specify predefined neuro-modules to be used during evolution, so that the evolving network structure can be influenced (see Doncieux et al. [2005]). Also, Doncieux et al. [2004b] proposed ModNet which supports connectivity patterns to influence the module connections in the initial generation.

20.1.2.4 Bootstrapping

One important problem with evolutionary algorithms and with neuroevolution of controller design in particular, is the bootstrapping of the evolution. The first few generations are especially important for NE algorithms. Promising candidates should be found within the range of an acceptable local optimum so that the gradient based search has a good chance of convergence. However, in the case where there are no promising candidates in the entire generation, then the search might take very long, or not succeed. This is because new candidates are preferably generated close to the best, but in this case they are bad performing individuals. This is particularly severe when there are low performing local optima in the fitness landscape. Then the evolution becomes more likely to get stuck in one of these undesired optima. Thus, it is important to start with a high diversity in the population and to keep it. One solution is to use shaping and iterative evolution (see above), because the simpler the experiments are, the smoother the fitness landscapes usually become. Finding a proper order for such iterative sub-tasks, however, is not always easy. To avoid fixed sequences of sub-tasks at all, Mouret et al. [2008] performed a multi-objective evolution on all sub-tasks at once, leading to an increased diversity. Another approach is network shaping, notably the pre-definition of network structures, so that it becomes more likely that random extensions of such an initial network results in promising candidates. In combination with suitable search space restriction measures, this increases the likeliness for good candidates.
While an effective strategy consists in using very large populations to guarantee many different randomly initialised candidates, the performance of the algorithm drops down. A good compromise is the use of interactive evolution algorithms, such as ENS3, where the population size and the probabilities for mutations can be adapted after the first few generations as a reaction to the observed progress. A similar effect can be achieved using an explicit initialisation generation in the algorithm, so that the population in the first generation is larger than in the remaining generations. The additional computational effort then is only required in the first generation to find suitable initial candidates with a broad search space sampling, whereas all following generations only evaluate the normal, lower number of individuals. Other strategies such as novelty search (see Section (20.1.2.2)) and behavioural distance measures (see Mouret et al. [2009a] [2009b]) can be used to actively search for differently behaving controllers with non-zero performance. These strategies help to create an early and long-lasting diversity in the population.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

20.1.3 Large networks

As the neural networks become larger, neuroevolution approaches try to increase the success rate of the evolutionary search with different strategies. There are three main strategies: structure reuse, incremental complexification and search space restriction.

20.1.3.1 Compressing patterns

A compositional pattern-producing network (CPPN) (see Stanley [2007]) is a way of compressing a pattern with regularities and symmetries into a relatively small set of genes. This is because natural brains exhibit numerous regular patterns (repeated motifs) such as in the receptive fields in the visual cortex. CPPNs can encode similar kinds of connectivity patterns. When they are used to generate the connectivity patterns of evolving ANNs, the resulting algorithm is called HyperNEAT (hypercube-based NEAT) (see D'Ambrosio et al. [2007], Gauci et al. [2007] Stanley et al. [2009]). This is because CPNN can be conceived as painting the inside of a hypercube representing the connectivity of an ANN. HyperNEAT does not evolve neural networks directly, but evolves function networks (CPPNs) that generate the weights of a fixed network topology, resulting in often repetitive, symmetric structures similar to neuro-modules. Fernando et al. [2016] proposed the differentiable pattern-producing networks (DPPNs) for optimising the weights of a denoising autoencoder. It is a differentiable version of the compositional pattern-producing networks (CPPNs).

20.1.3.2 Structure reuse

Structure reuse means that a single neural structure encoded directly or indirectly in the genome, can be reused in different parts of the network. It often relates to modular structures, or neuro-modules (Pasemann [1995], Auda et al. [1999]). The main advantage of reusing sub-networks is the possibility of developing functional building blocks, which can be reused in multiple parts of the network, without having to be reinvented by evolution multiple times. An example in biological systems are the cortical columns in the brain of mammals (see Horton et al. [2005]). Modular sub-networks are explicitly addressed in algorithms like Modular NEAT (Reisinger et al. [2004]), ModNet (Doncieux et al. [2004a]), CoSyNE (Gomez et al. [2008]), Cellular Encoding (Gruau [1994] [1995]) and its derivatives, ENSO (Valsalam et al. [2009], Valsalam [2010]), ESP (Gomez et al. [1997]), in SGOCE (see Meyer et al. 2003). Another approach to reuse structures is the generation of networks with CPPNs and the related evolution method HyperNEAT.

20.1.3.3 Incremental complexification

Another common strategy to reduce the search space is the incremental complexification of networks, as done in NEAT and its derivatives (see Stanley [2004]), ESP (Gomez [1997]), ModNet (Doncieux et al. [2004a]), ENS3 (Dieckmann [1995], Pasemann et al. [1999], Hulse et al. [2004]), NEATfields (Inden et al. [2010]) and most developmental approaches. Evolution is started with small or minimal networks, for instance only including the sensor and motor neurons, and explores this search space until no further improvement is observed. Then, the search space is repeatedly extended by adding additional neurons and synapses, and the new search space is explored. The assumption is that if solutions with simple network structures exist, then it would be beneficial to find them first, because they are expected to evolve faster than complex networks with the same functionality. This strategy has been shown to accelerate the evolution process in a number of experiments (see Stanley [2004]).

20.1.3.4 Search space restriction

Another way of reducing the structural search space is the consideration of symmetries and other topological peculiarities, like the positions of neurons, and the application of heuristics for the topology. For instance, developmental approaches with growing synapses usually implicitly implement the heuristics, that local, close-by neurons should have a higher probability to be connected than neurons further apart (see Nolfi et al. [1991], Meyer et al. [2003]), leading to less densely connected networks with locally connected groups of neurons.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Also strongly depending on the positions of the neurons, HyperNEAT can exploit topological neighbourhoods. Symmetries can be used in many experimental scenarios, because the bodies of most animats have one or more axes of symmetry. This can be reflected in the network topology and strongly reduces the search space.
ENSO (see Valsalam et al. [2009]) is an example of an algorithm that systematically breaks down symmetries step by step, starting with a predefined, highly symmetric network, hereby exploring the more symmetric, and thus smaller, search spaces first. Another search space restrictions is the manual exclusion of certain neurons or synapses from being changed during evolution, as it is possible with the ENS3 algorithm.

20.1.4 NEAT algorithm

We have presented in Section (20.1.1) the neuroevolution framework and discussed some techniques devised to improve the search. We are now going to detail the NEAT algorithm (see Stanley [2004]). As previously discussed, NEAT combines the usual search for the appropriate network weights with complexification of the network structure and use historical markings to line up genes with the same origin, allowing disparate topologies to cross over in a meaningful way. At last, it separates each innovation into a different species to protect topological innovation that needs a few generations. We are going to discuss the implementation of these features.

20.1.4.1 Genetic encoding

In order to get flexible genetic encoding, the author let each genome includes a list of connection genes. See Figure (20.3). Each connection gene specifies : the in-node, the out-node, the weight of the connection, whether or not the connection gene is expressed (an enable bit), and an innovation number, which allows finding corresponding genes during crossover.
In NEAT, the mutation can change

1. the connection weights: They mutate with each connection either perturbed or not.
2. the network structures: These mutations form the basis of comlexification. It occurs in two ways, see Figure (20.4):
   (a) Add connection mutation: a single new connection gene is added connecting two previously unconnected nodes.
   (b) Add node mutation: an existing connection is split and the new node placed where the old connection used to be. The old connection is disabled and two new connections added to the genome. The connection between the first node in the chain and the new node is given a weight of one, and the connection between the new node and the last node in the chain is given the same weight as the connection being split. This splitting is performed with a sigmoid function introducing nonlinearity, which slightly changes the function. As a result, the new structure evolves around the old one, allowing the genomes to gradually get larger. Genomes of varying sizes will result, sometimes with different connections at the same positions.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 20.3: NEAT genotype to phenotype mapping

Figure 20.4: NEAT two types of structural mutation

20.1.4.2 Crossover

We have described in Section (20.1.1.3) the problems arising with crossover when evolving neural networks in a deceptive environments. NEAT uses the historical origin of each gene to determine exactly which genes match up between any individuals in the population. Two genes with the same historical origin represent the same structure (although possibly with different weights), since they were both derived from the same ancestral gene at some point

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

in the past. Consequently, in order to properly align and recombine any two disparate topologies in the population, the system only needs to keep track of the historical origin of each gene.
Whenever a new gene appears (through structural mutation), a global innovation number is incremented and assigned to that gene. The innovation numbers thus represent a chronology of every gene in the population. Note, by keeping a list of the innovations that occurred in the current generation, we ensure that when the same structure arises more than once through independent mutations in the same generation, each identical mutation is assigned the same innovation number. The innovation numbers allow the system to know exactly which genes match up with which. See Figure(20.5). Genes that do not match are either disjoint or excess, depending on whether they occur within or outside the range of the other parent's innovation numbers.

Figure 20.5: NEAT matching genomes for different topologies with innovation numbers

When crossing over, the genes with the same innovation numbers are lined up. The offspring is then formed in one of two ways:

1. In uniform crossover: matching genes are randomly chosen for the offspring genome.
2. In blended crossover: the connection weights of matching genes are averaged.

Historical markings allow NEAT to perform crossover without analysing the topologies. Genomes of different organisations and sizes stay compatible throughout evolution, and the variable-length genome problem is essentially avoided, allowing NEAT to complexify structure.
Nonetheless, it is difficult for a population of varying topologies to support new innovations that add structure to

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

existing networks. This is because smaller structures optimise faster than larger structures. Thus, adding nodes and connections usually initially decreases the fitness of the network, so that recently augmented structures have little hope of surviving more than one generation. The solution is to protect innovation by speciating the population.

20.1.4.3 Speciation

Speciation [1] is performed so that individuals compete primarily within their own niches instead of with the population at large. Thus, topological innovations are protected and have time to optimise their structure before they have to compete with other niches in the population. Further, species with smaller genomes survive as long as their fitness is competitive, ensuring that small networks are not replaced by larger ones unnecessarily.
It uses historical markings to divide the population into species based on how similar they are topologically. The distance δ between two network encodings is measured as a linear combination of the number of excess E, disjoint genes D, and the average weight differences of matching genes $\overline{W}$:

$$\delta = c_1 \frac{E}{N} + c_2 \frac{D}{N} + c_3 \overline{W}$$

where the coefficients c_i, $i = 1, 2, 3$, adjust the importance of the three factors. N is the number of genes in the larger genome, normalising the genes. The genomes are tested one at a time: if the distance of a genome to a randomly chosen member of the species is less than a compatibility threshold δ_t, the genome is placed into this species. In the case where a genome is not compatible with any existing species, a new species is created. We can make δ_t dynamic by raising its value if there are too many species with respect to a given target, and lower it if there are too few species. Each genome is placed into the first species from the previous generation where this condition is satisfied, so that no genome is in more than one species. Keeping the same set of species from one generation to the next allows NEAT to remove stagnant species, that is, species that have not improved for several generations.
NEAT uses explicit fitness sharing as the reproduction mechanism (see details in Section (19.1.3.6)), where organisms in the same species must share the fitness of their niche. Thus, a species cannot afford to become too big even if many of its organisms perform well. Therefore, any one species is unlikely to take over the entire population, which is crucial for speciated evolution to support a variety of topologies.
The adjusted fitness $f_i^{'}$ for organism i is calculated according to its distance δ from every other organism j in the population. It is given by

$$f_i^{'} = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))}$$

where $sh(\cdot)$ is the sharing function which is set to 0 when the distance $\delta(i,j)$ is above the threshold δ_t, and 1 otherwise. Thus, the sum $\sum_{j=1}^{n} sh(\delta(i,j))$ reduces the number of organism in the same species as organism i. Note, the species are already clustered by compatibility using the threshold δ_t. Every species is assigned a potentially different number of offspring in proportion to the sum of adjusted fitnesses $f_i^{'}$ of its member organisms.
The net effect of fitness sharing is summarised as follows: Let $\overline{F}_k$ be the average fitness of species k and $|P|$ be the size of the population. We let $\overline{F}_{tot} = \sum_k \overline{F}_b$ be the total of all species fitness averages. The number of offspring n_k assigned to species k is

$$n_k = \frac{\overline{F}_k}{\overline{F}_{tot}} |P|$$

Species reproduce by first eliminating the lowest performing members from the population. The entire population is then replaced by the offspring of the remaining individuals in each species.

[1] Speciation is the evolutionary process by which populations evolve to become distinct species.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

20.1.4.4 Complexification

The final goal of the system is to perform the search for a solution as efficiently as possible, which is achieved through complexification from a simple starting structure. As a result of speciation, NEAT begins with a uniform population of simple networks with no hidden nodes, differing only in their initial random weights. New structures are introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. In this way, NEAT searches through a minimal number of weight dimensions, significantly reducing the number of generations necessary to find a solution, and ensuring that networks become no more complex than necessary. Then, NEAT searches for the optimal topology by incrementally complexifying existing structure.

20.2 Hybrid intelligent modelling

Data mining (see Section (3.3)) and more recent machine-learning methodologies (see Chapter (5)) provide a range of general techniques for the classification, prediction, and optimisation of structured and unstructured data. All of these methods require the use of quantitative optimisation techniques known as stochastic optimisation algorithms, such as combinatorial optimisation, simulated annealing (SA), genetic algorithms (GA), or reinforced learning. We find GA, ANN and SVM algorithms among the most popular classifiers. The artificial neural network (ANN) is the most widely used technique for classification and prediction. It is an optimisation task where the result is to find optimal weight and bias set of the network that reduce an error function. It can be formulated as the minimisation of an error function in the space of connection weights (see Section (10.1)). However, ANN has many disadvantages and some of its limitations are follows:

- ANN has long trained time
- High computational cost
- Adjustment of weight is difficult
- Output quality of an ANN may be unpredictable

To overcome these disadvantages Neural Networks can be combined with other techniques. One way forward is to combine ANN with another algorithm such as evolutionary computing (EC) tools that can take care of a specific problem.
Hybridisation is a technique which combines two or more classifiers to improve the performance of the classifier. Many researchers proposed hybrid technique with an evolutionary algorithm (EA) (see Section (19.1)) to improve the performance of the classifier. For instance, GAs has been hybridised with many other classification algorithms (see Montana et al. [1989]). Alternatively, Ilonen et al. [2003] applied Differential Evolution (DE) (see Section (19.1.4)) to train the weights of neural networks, obtaining convergence to a global minimum.

20.2.1 ANN plus GA in finance

20.2.1.1 An overview

The underlying principles of genetic algorithm (GA) (see Section (19.1.3)) are to generate an initial population of chromosomes (search solutions) and then use selection and recombination operators to generate a new, more effective population which eventually will have the fittest chromosome (optimal value) among them. Some examples of combition between ANN and GA are Whitley et al. [1990] who used GA to optimise weighted connections and find a good architecture for neural network connections. Kim [2006] proposed a hybrid model of ANN with GA that performs instance selection to reduce dimensionality of data. The genetic algorithm is used to optimise the connection weights between layers of neural network and for selection of relevant instance. Due to the selected instances, the learning time is reduced and prediction performance is enhanced. Bai et al. [2006] presented an effective hybrid system on rough

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

set and neural network. This classifier is used for managing the large document that is growing on the internet. The system creates clusters for organising the documents. The rough set is used for feature reduction and Neural Network is used for classification. Vivekanandan et al. [2010] built a rule-based classifier by using the genetic algorithm. The proposed work tries to reduce the learning time by incremental learning. To do so the classifier reduces the cost of learning and thus making it scalable for large data sets. Large researches have been performed in applying genetic algorithms for classification purpose. For instance, Bhardwaj et al. [2015] proposed a model called GONN which hybridise genetic programming with neural network for multi class classification problem. The model simultaneously deals with structure and weight of the neural network, bringing more diversity of the GP population and reaching a solution faster with more generalised solutions. GAs has been also widely used for the discovery of classification rules.

20.2.1.2 Some examples

20.2.1.2.1 Feature selection Sangwan et al. [2015] proposed an integrated ANN and GA for predictive modelling and optimization of turning parameters to minimize surface roughness. Following the same approach, Inthachot et al. [2016] used GA to solve a feature selection problem to find effective subsets of input into ANN. Considering four input variables for each technical indicator, to speed up computation time, they used GA to devise a small number of effective subsets of input variables (see Figure (20.6)). Accuracy is used to determine chromosome selection as well as to measure the performance of the prediction model.

Figure 20.6: Steps of operation of ANN and GA hybrid intelligence

20.2.1.2.2 Weight selection Karimi et al. [2012] used GA to find a set of weights for connections to each node in an ANN model and determine correlation of density in nanofluids. Qiu et al. [2016] proposed to determine the optimal set of weights and biases to enhance the accuracy of an ANN model (backpropagation algorithm) by using genetic algorithms (GA). They reported improved prediction accuracy of the direction of stock price index movement with their optimised ANN model. They considered a three-layer ANN model where the output layer consists of only one neuron that represents the predicted direction of the daily stock market index. The signals are generated on a daily basis at time interval $\delta = 1$ day and the forecasting period is $d = 1$ day. GA technique is used to optimise the (initial) weights and biases of the ANN model. Then, the ANN model is trained by the BP algorithm using the determined weights and bias values. The steps are as follow:

- The data is first normalised (see Equation (12.1.1)), then all the weights and biases are encoded in a string and the initial population is generated. Each solution from the GA is called chromosome (or individual) describing the ANN with a certain set of weights and bias values.
- Train the ANN model using the BP algorithm and then evaluate each chromosome of the current population using a fitness function based on the MSE.
- Rank all the individuals using the fitness proportion method and select the individuals with a higher fitness value to pass on to the next generation directly.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Apply crossover and mutation to the current population and create new chromosomes. Evaluate the fitness value of the new chromosomes and insert these new chromosomes into the population to replace the worse individuals of the current population.
- Repeat the previous steps until the stop criterion is satisfied.

20.2.2 ANN plus DE in finance

The training (error) of artificial neural networks (ANNs) depends on the adaptation of free network parameters, such as the weight values and the bias. Thus, it is an optimisation task where the result is to find optimal weight and bias set of the network that reduce an error function. It can be formulated as the minimisation of an error function in the space of connection weights (see Section (10.1.1)). The methods for training ANNs are backpropagation (BP), Levenberg-Marquadt (LM), Quasi-Newton (QN). In general, the error backpropagation method (EBP), based on gradient method, is preferred. However, the objective function describing the artificial neural networks training problem is a multi-modal function, so that the algorithms based on gradient methods can easily be stuck in local extremes. To avoid this problem one can use a global search optimisation, such as simulating annealing (SA) algorithm (see Aarst et al. [1989]), Particle Swarm Optimization (PSO) and Evolutionary Algorithm (EA) (see Michalewicz [1996]). For instance, Junyou [2007] used PSO-trained neural networks to forecast the stock price.
We have seen above that when training the ANN by using GA method (GANN), the weight coefficients of neural network are encoded in chromosome, and selection, cross-over and mutation operators are used to minimise the error. However, GANN suffers from early convergence. Even though GA has diversity in its population, it lacks of convergence speed towards global optimia. On the other hand, PSO has faster convergence speed than that of GA but it lacks of diversity in population.

We focus on differential evolution (DE) (see Section (19.1.4)), which has the following advantages: the possibility of finding the global minimum of a multi-modal function regardless of initial values of its parameters, quick convergence and a small number of parameters to set up at the start of the algorithm operation.
Ilonen et al. [2003] applied Differential Evolution to train the weights of neural networks, obtaining convergence to a global minimum but in a very long time. To do so, the weights of all neurons are stored in a real valued solution vector X. The algorithm is used to minimise the sum of squared errors (SSE) or a similar criterion function. The evaluation of this function requires iterating through all elements of the training set T and summing all the partial results (squared errors in the case of SSE) obtained for all the elements of T. Note, different choices of behavioural parameters cause it to perform worse or better than BP on particular problems and the selection of good parameters is a challenge. One way forward is to try and make the DE parameters automatically adapt to new problems during optimisation, hence alleviating the need for the practitioner to select the parameters by hand. Since some meta parameters are kept fix throughout the entire evolution process, one must tune value of these control parameters. One approach is to use self-adaptive strategy for controlling these parameters (see Brest et al. [2006]). Slowik et al. [2008] applied DE technique to train a feed-forward flat ANN to classification of parity-p problem by introducing the adaptive selection of control parameters to the algorithm. The method proposed is called DE-ANNT (Differential Evolution ? Artificial Neural Network Training). As a result, only one parameter is set at the start of proposed algorithm. Similarly, Bui et al. [2015] use self-adaptive strategy to control the DE parameters. Rather than considering self-adaptive parameters, Pedersen et al. [2008b] proposed an overlaying optimisation method known as meta-optimisation.

Even though EA simultaneously process a population of problem solutions, it comes at the expense of very high computational complexity. To remedy this problem, Fan et al. [2003] introduced Trigonometric DE (TDE) algorithm and applied to train the ANN as a test case for their proposed algorithm. Bandurski et al. [2009] proposed to combine differential evolution algorithm with a gradient-based approach. This hybridisation comes in a number of ways, some of them are as follows:

1. One solution is for an EA to be used to locate a promising region of the weight space, and then use a gradient

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

descent method to fine-tune the best solution (or all the solutions from the population) obtained by the EA (see Sherrod [2008]).

2. Another solution is to apply Lamarckian evolution (see Ross [1999]) to ANN and let the gradient descent procedure be incorporated into an EA as a new search operator. This operator is applied to the population members in each EA iteration, in addition to standard operators such as mutation and crossover (see Cortez et al. [2002]).

The authors applied the conjugate gradient (CG) algorithm (see Hestenes et al. [1952]) to each candidate solution $U_{i,G}$ (Trial vector in Appendix (19.1.4.1)) before the computation of its fitness. The number of CG iterations is set by the user and remains constant throughout the entire experiment. The results obtained were significantly faster than the original version of DE.
Similarly, to keep a reasonable balance between convergence speed and the capability of global search, Mingguange et al. [2009] combined the BP and DE algorithm to optimise the weights and threshold value adjustments of ANN.

Si et al. [2012] used DE with global and local neighbourhood based mutation (DEGL) algorithm to search the synaptic weight coefficients of neural network and to minimise the learning error in the error surface. It is a modification of DE where both global and local neighbourhood-based mutation operator is combined to create donor vector.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 21

Solving optimisation problems with ML

21.1 Solving constrained optimisation problems

In general, optimisation problems that can not be solved with an exact algorithm are solved by applying a global search optimisation such as genetic algorithms, particle swarm optimisation, simulated annealing, ant colony optimisation etc. Some authors proposed the use of neural networks (NN) to resolve optimisation problems in those cases where the use of linear programming or Lagrange multipliers (see Section (18.5)) is not feasible.

21.1.1 The method

21.1.1.1 A short review

Artificial neural networks (ANN) have been used to obtain solution of constrained optimisation problems (see Chapter (18)). For instance, Chua et al. [1984] developed the canonical non-linear programming circuit, using the Kuhn-Tucker conditions from mathematical programming theory (see Section (18.5)). Hopfield et al. [1985] [1987] published some results on how to go about using neural networks to solve optimisation problems. The Travelling Salesman Problem was the first problem formulated in terms of neural network. The authors modified the energy function of the network to make it equivalent to TSP objective and used Lagrange multipliers to penalise the violations of the constraints. The approach was then used to demonstrate how circuits of simple unit can solve hard problems. Tank et al. [1986] developed their optimisation network for solving linear programming problems. Smith et al. [1989] discussed some practical design problems of the Tank and Hopfield (TH) network along with its stability properties. Kennedy et al. [1988] extended the results of TH to more general non-linear programming problems. Further, they showed that the network introduced by TH was a special case of the canonical non-linear programming network proposed Chua et al. [1984] with capacitors added to account for the dynamic behaviour of the circuit. Gee [1993] overcame some of the limitations of the TH network. In parallel to the development of Hopfield networks to solve TSP we can list the deformable template models, the Elastic Nets (see Durbin et al. [1987]), and the application of Self Organizing Map (see Fort [1988], Kohonen [1990]). Lillo et al. [1993a] analysed the dynamics of the canonical nonlinear programming circuit and showed that it was a gradient system minimising an unconstrained energy function that can be viewed as a penalty method approximation of the original problem. Thus, all of these methods (Hopfield-type networks) use the penalty function method (see Section (18.7.2)) whereby a constrained optimisation problem is approximated by an unconstrained optimisation problem. This is due to the network property of reducing their energy function during evolution, leading to a local or global minimum. Some authors proposed an exact penalty function approach and presented a neural optimisation network for solving constrained optimisation problems (see Lillo et al. [1993b], Mladenov et al. [1999]). The parallel nature of some neural network models seems to be very promising in reducing computation time (see Cichocki et al. [1993], Takefuji et al. [1996]). However, the problem of local minima is the main limitation of the approach and several techniques have been proposed to overcome it, such as stochastic

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

networks, simulated annealing, and other global optimisation methods. Villarrubia et al. [2017] proposed to apply a multilayer perceptron to approximate the objective functions. The same process could be followed in the restrictions. The objective function is approximated with a non-linear regression with the objective of obtaining a new function that facilitates the solution of the optimisation problem. The activation function of the neural network must be selected so that the derivative of the transformed objective functions should be polynomial.

21.1.1.2 Mapping optimisation problems to networks

Since a Hopfield network (see Section (5.1.2)) performs as a minimiser of its energy function, if the optimisation problem can be coded as an energy function, then a network that corresponds to this energy function can be used to minimise (locally or globally) the function and thus provide an optimal or near-optimal solution. Thus, the procedure starts with the construction of the energy function and then the parameters of the network (number of units, weights of connection, thresholds, update policy or even the dynamics) are adjusted to reflect the problem. Then the network is initialised to some initial state and is let to run until it comes to equilibrium from where a solution can be drawn. However, in order to construct the appropriate energy function the latter must be quadratic to satisfy Equation (5.1.3). The most common approach is the penalty (or cost) function approach (see Section (18.7)). The energy function is initialised to the objective function of the problem and for each constraint a penalty term is added. Thus, the problem from a constrained optimisation form is reduced to an unconstrained minimisation problem.
As soon as the energy function has been constructed, the elements of the network (number of units, weights, thresholds) can be derived. For each unit i the energy difference ΔE_i is calculated (derivatives can be used for this purpose) and it is transformed into a form similar to Equation (5.1.4). By analogy, the coefficient of v_j will give the weight w_{ij}, whereas the constant terms will give the threshold θ_i. Note, as the problem size grows, the network size (units and connections) can be intractable large and inapplicable to practical domains. The limitations are due to storage and computing power requirements, but several techniques for parallel implementations have been proposed. The initialisation of the network state is a problem since if it is initialised to a state corresponding to a possible solution, it will be stuck as all the constraints are satisfied, although this solution may not be optimal. Such states corresponds to local minima and are not desirable as initial states. Moreover, different initialisation may lead to different solutions, with different quality and/or computation time. One solution is to consider a random state for initialisation.

21.1.2 Some optimisation models

We briefly present some simple networks used for optimisation problems.

21.1.2.1 Discrete synchronous Hopfield network

In the discrete synchronous Hopfield network, the dynamics of the system are given by Equation (5.1.1) and all the units are updated in parallel. The network is not guaranteed to decrease at each step and since the system may fail to come into equilibrium a solution is not always derived. However, it has the advantage of being fully parallelisable.

21.1.2.2 Discrete asynchronous Hopfield network

This model is similar to the previous one with the difference that the units are updated sequentially. The network will eventually come into equilibrium, from where a solution can be drawn. However, the system can be easily trapped into a local minimum and is highly dependent on the initial state. Further, it can not be parallelised since it is strictly sequential.

21.1.2.3 Analog Hopfield network

In the analog Hopfield network, the dynamics of the system are given by Equation (5.1.1). Note, synchronous, asynchronous or continuous updating can be applied. Although the output of such networks is continuous they can be used to solve discrete optimisation problems. In the equilibrium state the outputs closer to 1 are taken as 1 and the output

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

closer to 0 (-1) are taken as 0 (-1). When using the sigmoid function, if we start the network at the temperature T where we want to measure the outputs, it may take a long time to come to equilibrium. In general, the simulated annealing technique is applied. We start the network at a relatively high temperature and gradually we cool it down. The potential disadvantage of this network is that it may be trapped in local minima inside the hypercube without reaching any of the corners.

21.1.2.4 Boltzmann machine

The operation of the Boltzmann machine integrates the dynamics of the discrete asynchronous Hopfield model with the simulated annealing technique. At each step t a unit i of the network is selected randomly and the energy difference ΔE_i that will be caused by a change of its state is calculated using Equation (5.1.4). If ΔE_i is negative (the energy is decreased) the change is definitely accepted, otherwise it is accepted with probability $P_{B,i}(t)$ that depends on the quantity $e^{\frac{\Delta E_i}{T_B}}$ where T_B is a temperature that decreases according to some annealing schedule. In general, the Metropolis criterion is used as the acceptance criterion, which is given by

$$P_{B,i}(t) = \begin{cases} 1 \text{ if } \Delta E_i(t) < 0 \\ \frac{1}{1+e^{\frac{\Delta E_i}{T_B}}} \text{ if } \Delta E_i(t) \geqslant 0 \end{cases}$$

The logarithmic schedule can be used to decrease the temperature:

$$T_B(t) = \frac{T_B(t-1)}{1 + t\log(1+r)}$$

where r is a parameter that adjusts the speed of the schedule.
The Boltzmann machine can be very effective when used with the appropriate annealling schedule and is able to perform a wide exploration of the problem state space. However, it is strictly sequential and can not be parallelised. Several attempts at parallelising its operation (group updates) were proposed, all having to cope with some kind of trade-off between the solution quality and the actual speed-up.

21.1.2.5 Cauchy machine

The Cauchy machine extends the discrete synchronous Hopfield model. The behaviour of a unit i at each time step t is given by

$$v_i(t) = \Theta(u_i(t))$$

where u_i is the activation in Equation (5.1.2). During the operation, the activation is updated using the following motion equation (gradient descent dynamics)

$$\frac{du_i(t)}{dt} = -\frac{\partial E(v)}{\partial v_i}$$

For simulation purposes, we use the first-order approximation for the above dynamics, getting

$$\frac{\Delta u_i}{\Delta t} = -\frac{\Delta E_i}{\Delta v_i} \text{ and } u_i(t+\Delta t) = u_i(t) + \Delta u_i$$

Note, the activation plays the role of an accumulator, a sort of a memory that stores the cumulative activation of the unit during the network's operation time. Hence, the probability of two units to change state simultaneously is reduced significantly, so that oscillation phenomena are avoided. Consequently, the system will eventually come to equilibrium. This is strengthened also by the fact that each unit follows the above gradient descent dynamics. When the energy function is given by Equation (5.1.3), then using Equation (5.1.4) and the above gradient descent dynamics we can derive the motion equation for each unit i as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\frac{\Delta u_i}{\Delta t} = \sum_{j=1}^{n} w_{ij} v_j + \theta_i \text{ and } u_i(t+\Delta t) = u_i(t) + \big(\sum_{j=1}^{n} w_{ij} v_j + \theta_i\big)\Delta t$$

A state vector v constitutes an equilibrium state for the network if for all i the following condition is satisfied

$$\big(v_i = 1 \text{ and } \Delta u_i \geqslant 0\big) \text{ or } \big(v_i = 0 \text{ and } \Delta u_i \leqslant 0\big)$$

In order to provide the system with stochastic hill-climbing capabilities, the Distributed Cauchy Machine has been developed, where Cauchy color noise is added in the updating procedure. The output of unit i at each time step t is stochastically updated with probability $P_{C,i}(t)$ which depends on the Cauchy distribution:

$$P_{C,i}(t) = \begin{cases} s_i(t) \text{ if } v_i(t) = 0 \\ 1 - s_i(t) \text{ if } v_i(t) = 1 \end{cases}$$

where

$$s_i(t) = P(v_i(t) = 1) = \frac{1}{2} + \frac{1}{\pi} arctan\big(\frac{u_i(t)}{T_C(t)}\big)$$

The virtual temperature $T_C(t)$ is usually given by a fast annealing schedule:

$$T_C(t) = \frac{T_{C,0}}{1 + \beta t}$$

where $T_{C,0}$ is the initial temperature and β is real parameter in the range $[0, 1]$ controlling the speed of the schedule. In the extreme case where $T_C = 0$ the network becomes deterministic.
In general, the Cauchy machine provides solutions of lower quality than the Boltzmann machine, but it has the advantage of being fully parallelisable. Further, since the activation is cumulative, changes at the state of a unit can be done only after many time steps, for the activation must change sign. The lack of this flexibility becomes more obvious when the system has operated for a long time and the activations have large absolute values.

21.1.2.6 Hybrid Scheme

Papageorgiou et al. [1998] presented the hybrid update scheme as an attempt at combining both the advantages of the Boltzmann and the Cauchy machine. It extends the synchronous discrete Hopfield model and the stochastic update rule is based on a convex combination of the Boltzmann and Cauchy machine update rules. At each time step t, the probability of accepting a state change concerning unit i is given by

$$P_{H,i}(t) = \alpha P_{C,i}(t) + (1 - \alpha) P_{B,i}(t)$$

where α is a control parameter in the range $[0, 1]$. A large value of α will result to a typical Cauchy machine, whereas a small value will result to a synchronous Boltzmann machine destroying the convergence property. The two temperatures need not be equal and a good choice would be to update $T_C(t)$ according to some annealing schedule and then take $T_B(t) = \lambda T_C(t)$ where λ is an adjustable parameter. Even though the above stochastic update rule accepts changes suggested by the energy difference (through $P_{B,i}(t)$), such changes should be reflected on the activation $u_i(t)$, otherwise at the next time step the unit would return to the previous value, since it is still suggested by the activation. The following rule ensures that this will not happen:

$$\begin{aligned} \text{if} \quad & \big(u_i(t+\Delta t) <> u_i(t)\big) \text{ and } \big(P_{C,i}(t) < 0.25\big) \text{ and } \big(P_{B,i}(t) > 0.75\big) \\ \text{then} \quad & \big(u_i(t+\Delta t) = -u_i(t)\big) \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Lagoudakis [1997] showed that $\left(P_{B,i}(t) > 0.75\right)$ was necessary for the network to converge. Experimental results showed that the solutions produced by the hybrid scheme were similar to the solutions produced by the Boltzmann machine but with larger convergence time, similar to that of the Cauchy machine. However, the hybrid scheme is fully parallelisable, so that the parallel implementation provides solutions similar to that of the Boltzmann machine but in substantially less time.

21.1.3 Approximation with non-linear regression

21.1.3.1 The method

The main idea is to approximate the objective function f with some heuristics methods. Approximation functions are usually defined around a point, which would make it possible to use polynomials to approximate functions by applying the Taylor theorem. Based on this idea, one can solve non-linear optimisation problems by applying Taylor non-linear functions. For instance, Nanculef et al. [2014] applied this method in Frank-Wolfe algorithms, allowing for the linearisation of the objective functions by applying derivatives in a point to calculate the straight line, plane or hyperplane crosses through that point. The solutions are calculated iteratively with a new hyperplane for each iteration. Note, the MAP (Method of Approximation Programming) is a generalisation of the Frank-Wolfe algorithm, which permits to linearise the restrictions.
Villarrubia et al. [2017] proposed to generalise this approach by using the universal approximation theorem (see Theorem (5.1.1) and approximate the objective function with neural networks. That is, rather than calculating a new approximation for each tentative solution, they chose to infer the approximation by applying neural networks. Given a known objective function f, the system generates a dataset in the domain of the variables to train a neural network. The objective function of the optimisation problem is redefined with the multilayer perceptron that transforms the function, making it possible to generate a polynomial equation to resolve the optimisation problem. Finally, when the new objective function is calculated another solution (such as the Lagrangian method) can be applied to resolve the problem.
To define a neural network, it is necessary to establish parameters, such as the connections, number of layers, activation functions, propagation rules etc. In the case of the multilayer perceptron, the authors considered two different stages: the learning stage, and the prediction process. In both stages, the number of layers and activation functions have to be the same.

21.1.3.2 Defining the network

Following the Hopfield networks (see Section (5.1.2)) and multilayer networks (see Section (5.2)), the propagation rule is the weighted sum (see activation in Equation (5.1.2)) defined by

$$u_j(t) = \sum_{i=1}^{n} w_{ij} x_i(t) + \theta_j$$

where w_{ij} is the weight connecting neuron i in the input layer with neuron j in the hidden layer , x_i is the output from neuron i in the input layer, n is the number of neurons in the input layers, t is the pattern, and θ_j is the bias.
We now need to select the activation function Φ of the network such that that the derivatives of the transformed objective functions are polynomial. This is because it will simplify calculations when using the Lagrangian method to solve the optimisation problem. If the function Φ is linear, the output of neuron j is a linear combination of the neurons in the input layer and, consequently, y_j is a linear function. Therefore, if the activation function is the identity, the net output would correspond to the output of the neuron. Thus, if neuron j in the hidden layer has the activation function Φ , the output is given by

$$y_j(t) = \Phi(u_j(t))$$

which is equivalent to a non-linear regression.
Since the multilayer perceptron has three layers, we apply the propagation rule twice in order to transmit the value

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

from the input layer to the neurons in the output layer. If neuron k in the output layer has the activation function Φ_{out}, the output is given by

$$y_k(t) = \Phi_{out}\big(\sum_{j=1}^{m} w_{jk} y_j(t) + \theta_k\big)$$

where m is the number of neurons in the hidden layer. In the special case where Φ_{out} is the identity function, the output simplifies to

$$y_k(t) = \sum_{j=1}^{m} w_{jk} y_j(t) + \theta_k$$

Replacing $y_j(t)$ with its value defined above, the output in neuron k is defined as

$$y_k(t) = \sum_{j=1}^{m} w_{jk} \Phi(u_j(t)) + \theta_k \tag{21.1.1}$$

Using the universal approximation theorem (see Theorem (5.1.1)), we can deduce that there exists $\epsilon > 0$ such that

$$|y_k(t) - f| < \epsilon$$

In the special case where the function Φ is the identity, the output in neuron k from the output layer is calculated as a linear combination of the inputs, so that the function is linear. Thus, if we train the multilayer perceptron with an identity activation function, it would be possible to make an approximation of the trained function. In the case where an optimisation problem has a non-linear objective function, it would then be possible to redefine the function according to Equation (21.1.1) so that it would be linear. However, with this activation function we can not prevent the approximation functions from becoming hyperplane, so that the activation function Φ can not be linear. Another choice would be to consider the arctan activation function because the equation of the derivative [1] is simpler than the expression of the sigmoidal function.

21.1.3.3 The Lagrangian method

In order to solve the optimisation problem defined in Equation (18.1.2), we can approximate the objective function f with the neural network defined in Equation (21.1.1), getting

$$f(x) \approx \sum_{j=1}^{m} w_{jk} \Phi(u_j(t)) + \theta_k \text{ such that } g_i(x) \leqslant 0 \,,\, i = 1, ..., m$$

We can then use the Lagrangian approach (see Section (18.5)) to solve this optimisation problem. In that case the Lagrangian in Equation (18.5.3) becomes

$$L(x, \lambda) = \sum_{j=1}^{m} w_{jk} \Phi(u_j(t)) + \theta_k + \sum_{i=1}^{m} \lambda_i \cdot g_i(x)$$

with the Karush-Kuhn-Tucker (KKT) conditions

$$\begin{aligned} \frac{\partial L}{\partial x_i} &= 0 \,,\, i = 1, ..., n \\ \frac{\partial L}{\partial \lambda_j} &= 0 \,,\, j = 1, ..., m \end{aligned}$$

[1] $arctan(f) = \frac{f'}{1+f^2}$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The stationarity condition can be written as

$$\forall j \quad \frac{\partial f(x)}{\partial x_j} \pm \sum_{i=1}^{m} \lambda_i \frac{\partial g_i(x)}{\partial x_j} \leqslant 0$$

and the complementary condition is

$$\forall i : \lambda_i g_i(x) = 0$$

We clearly see that we need to use an activation function Φ which is analytically differentiable to simplify the derivatives in order to calculate the solution more easily. For restrictions with inequality, it is possible to introduce a threshold based on the training carried out in the neural network. For the threshold to be lower, it would be best to have previously trained the neural network with values in the variables around their definition. That is, in a problem with restrictions given by

$$\begin{aligned} a_i &\leqslant x_i \leqslant b_i \\ x_i &\geqslant a_i \\ x_i &\leqslant b_i \end{aligned}$$

we should generate a dataset for the training phase that matches the restrictions for the variable x_i. The lower the difference among consecutive values, the lower the error to define the threshold. The authors got the restriction

$$g_s(x) \leqslant 0$$

and given Equation (21.1.1), they got

$$\sum_{j=1}^{m} w_{jk} \Phi(u_j(t)) + \theta_k + \mu_s = 0$$

Finally, the optimisation problem becomes

$$f(x) = \sum_{j=1}^{m^1} w_{jk}^1 \Phi\big(\sum_{i=1}^{n^1} w_{ij}^1 x_i^1(t) + \theta_j^1\big) + \theta_k^1$$

such that

$$\begin{aligned} \sum_{j=1}^{m^1} w_{jk}^1 \Phi\big(\sum_{i=1}^{n^1} w_{ij}^1 x_i^1(t) + \theta_j^1\big) + \theta_k^1 + \mu_s^1 &= 0 \\ &\vdots \\ \sum_{j=1}^{m^n} w_{jk}^n \Phi\big(\sum_{i=1}^{n^n} w_{ij}^n x_i^n(t) + \theta_j^n\big) + \theta_k^n + \mu_s^n &= 0 \end{aligned}$$

21.2 Solving combinatorial optimisation problems

We have presented continuous optimisation problems in Chapter (18). We are now going to briefly introduce discrete optimisation problems and focus on combinatorial optimisation. In an optimisation problem with discrete variables, we are looking for an object such as an integer, permutation or graph from a finite (or possibly countably infinite)

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

set. There is an objective function to be minimised, with a very large discrete configuration space. We defined in Section (18.1.1) the optimisation problem. In the case of discrete optimisation the set $\mathcal{S}$ is a discrete, countable set. In contrast to continuous sets and continuous functions, the concept of a local minimum can not be defined unless a suitable metric (or distance between the solutions) is defined. Such a metric is highly problem dependent and can be used to define a neighbourhood structure in the solution space. Optimality is defined with respect to some criterion function, which is to be minimised or maximised such as:

- minimisation: cost, distance, length of a traversal, weight, processing time, material, energy consumption, number of objects.
- maximisation: profit, value, output, return, yield, utility, efficiency, capacity, number of objects.

The solutions may be combinatorial structures like arrangements, sequences, combinations, choices of objects, sequences, subsets, subgraphs, chains, routes in a network, assignments, schedules of jobs.

Some examples of combinatorial optimisation problems are shortest paths and shortest path trees, flows and circulations, spanning trees, matching and matroid problems. In graph theory, the shortest path problem consists in finding a path between two vertices in a graph such that the sum of the weights of its constituent edges is minimised. In this section, we will focus on the travelling salesman problem (TSP), which consists in finding the shortest path going through every vertex exactly once, and returning to the start.
In practice, TSP solvers rely on handcrafted heuristics that guide their search procedures to find competitive (and even optimal) tours efficiently. Those heuristics describe how to navigate the space of feasible solutions in an efficient manner (see Applegate et al. [2006] [2011]). More generic solvers, rely on a combination of local search algorithms and metaheuristics (see Google 2016). However, if the problem statement slightly changes, the heuristics need to be revised.

We briefly describe these standard methods and discuss how neural networks (NNs) and reinforcement learning have been used to solve the travelling salesman problem.

21.2.1 Standard methods

We have presented discrete optimisation in Section (18.1.3), introduced the travelling salesman problem (TSP), and discussed some solution methods in Section (18.1.3.2). We are now going to formalise the combinatorial optimisation problem and present some well known solutions.

21.2.1.1 Formalising the problem

A combinatorial optimisation problem A is a quadruple (I, f, m, g) where

- I is a set of instances.
- given an instance $x \in I$, $\mathcal{S}(x)$ is the set of feasible solutions.
- given an instance x and a feasible solution y of x, $m(x, y)$ denotes the measure of y which is usually a positive real.
- f is the goal function, and is either $\min$ or $\max$.

The goal is then to find, for some instance x, an optimal solution, that is, a feasible solution y with

$$m(x, y) = f\big(m(x, y^{'})|y^{'} \in \mathcal{S}(x)\big)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

For each combinatorial optimisation problem, there is a corresponding decision problem that asks whether there is a feasible solution for some particular measure m_0.
In the field of approximation algorithms, algorithms are designed to find near-optimal solutions to hard problems. The usual decision version is then an inadequate definition of the problem since it only specifies acceptable solutions. Nonetheless, even though we could introduce suitable decision problems, the problem is more naturally characterised as an optimisation problem.
An algorithm is said to be of polynomial time if its running time is upper bounded by a polynomial expression in the size of the input for the algorithm, that is, $T(n) = O(n^k)$ for some positive constant k. Problems for which a deterministic polynomial time algorithm exists belong to the complexity class P, which is central in the field of computational complexity theory.
An NP-optimisation problem (NPO) is a combinatorial optimisation problem with the following additional conditions:

- The size of every feasible solution $y \in \mathcal{S}(x)$ is polynomially bounded in the size of the given instance x.
- The languages $\{x|x \in I\}$ and $\{(x, y)|y \in \mathcal{S}(x)\}$ can be recognised in polynomial time.
- m is polynomial-time computable.

This implies that the corresponding decision problem is in NP. Here, the polynomials are functions of the size of the respective functions inputs, not the size of some implicit set of input instances. A problem is additionally called a P-optimisation (PO) problem, if there exists an algorithm which finds optimal solutions in polynomial time. Often, when dealing with the class NPO, one is interested in optimisation problems for which the decision versions are NP-complete.

21.2.1.2 Some solutions

We defined in Section (18.6.2.2), the integer linear program (ILP). There exist several different strategies for solving or approximating ILPs. In general, these strategies first attempt at solving relaxation of the original problem, which are obtained by dropping some of the inequalities. For instance, the LP-relaxation is the LP obtained by dropping the integer condition. In the case of combinatorial optimisation problems we restrict the variables to be binary. Some simple solutions were proposed to solve ILP such as

- Cutting: adding extra linear inequalities. We consider relaxation of the problem and repeatedly cuts away parts of the polytope by adding new constraints in the hope of obtaining an integer solution. To find cutting planes we need to solve the separation problem. That is, find a violated, valid inequality of the ILP.
- Branching: dividing into sub-problems. An enumeration tree of all possible variable settings is partially traversed, computing local upper bounds and global lower bounds, which are used to avoid parts of the tree that can not produce the optimal value.

The cutting plane method by Gomory (see Cornuejols [2007]) is as follows:

1. Solve the initial LP relaxation with a variant of the simplex method.
2. Let x^* be the current solution vector. If x^* is integral, stop.
3. Let x_k^* be fractional. There is is a row of the simplex tableau of the form

$$x_k + \sum_{i \in NB} c_i x_i = x_k^*$$

Generate the cutting plane

$$\sum_{i \in NB} f(c_i) x_i \geqslant f(x_k^*)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

4. Add the cutting plane to the LP and re-optimise via a dual simplex pivot. Return to [2].

If the solution x of the LP-relaxation is integral, it corresponds to a feasible incidence vector representing an optimal solution. Otherwise, we search for a valid inequality that cuts off the solution x. This algorithm is finitely convergent given a certain rule for choosing the fractional variable in step [3]. However, a very large number of cutting planes are typically needed, and feasible solution is only obtained at the end of the process.
Similarly, we need to explore a large number of sub-problems in the branch-and-bound method. Further, performing bread-first search requires a huge amount of memory. Performing depth-first leads to spending long time exploring dead-ends.
More generally, we write the linear programming relaxation as

$$\min\{cx : \ Ax \leqslant b \, , x \in \mathbb{R}_+^N\}$$

its feasible region is

$$P = \{x \in \mathbb{R}_+^N : \ Ax \leqslant b\}$$

the set of integer solution is

$$\{x \in \mathbb{Z}_+^N : \ Ax \leqslant b\}$$

and the integral hull is

$$P_I = conv\{x \in \mathbb{Z}_+^N : \ Ax \leqslant b\}$$

The strongest cutting planes are the one defining facets of P_I.
In the case of the symmetric TSP (STSP) (see details in Section (18.6.3)), for each edge $e \in E$, x_e is a binary variable taking the value 1 if e is in the tour and 0 otherwise. The vectors x representing tours satisfy

$$i \in V \, , \sum_{e \in \delta(i)} x_e = 2 \text{ and } W \subset V \, , \sum_{e \in \delta(W)} x_e \geqslant 2 \ (SECs) \text{ and } x \in \{0,1\}^{|E|}$$

The integral hull is called the symmetric travelling salesman (STS) polytope. The trivial bounds $0 \leqslant x_e \leqslant 1$ induce facets and the SECs induce an exponential number of facets. To get a complete description for general N we need $NP = co - NP$.
In order to use strong valid inequalities in a cutting plane algorithm, we need to detect when they are violated by the solution to a given LP relaxation. That is, given a vector $x^{* \in P \setminus P_I}$, we want to find an inequality (facet-inducing) which separates x^* from P_I. This problem is called the separation problem (see Grotschel et al. [1988]).

Theorem 21.2.1 *For any proper class of polyhedra, the optimisation problem is polynomially solvable if and only if the separation problem is polynomially solvable.*

However, if the ILP is NP-hard:

- We do not know all the facets.
- Even the known valid inequalities / facets are usually exponential.
- Testing whether there is a cut separating x^* from P_I is co-NP complete.

Nonetheless, for some particular classes of inequalities, we have algorithms for detecting if an inequality in that class is violated. Thus, we can conclude that an exact separation algorithm for a given class of inequalities is a procedure taking the vector $x^* \in P$ as input, and either outputs one or more inequalities in the class violated by x^*, or proves that none exists.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

This is similar for heuristic separation algorithms except that it outputs either one or more inequalities in the class violated by x^*, or a failure message. In the case of the TSP there are exact separation algorithms for the SECs, the 2-matching inequalities, and certain variants of the comb inequalities.
It led to the strong cutting plane algorithm:

1. Solve an initial LP relaxation.
2. Let x^* be the solution. If it is integer and feasible, output the optimal solution and stop.
3. Attempt to find some facet-inducing inequalities violated by x^*. If none are found, output the final lower bound and stop.
4. Add the facet-inducing inequalities to the LP. Resolve the LP and go to $[2]$.

Later, some authors called the separation algorithms at every node of the branch-and-bound tree. For instance, Padberg et al. [1991] devised the full paradigm called branch-and-cut:

1. LP-based branch-and-bound plus
2. Separation called at each node of tree
3. Cutting planes globally available via cut pool
4. Reduced-cost fixing
5. LP-based primal heuristic

This algorithm is very effective for the TSP (see Applegate et al. [2011]). However, it was not that successful on tightly constrained machine scheduling or vehicle routing problems, or on some graph theory problems.

21.2.2 Solving the TSP with ML

21.2.2.1 A review

Motivated by the recent advancements in sequence-to-sequence (seq2seq) learning (see Sutskever et al. [2014]), neural networks are again the subject of study for optimisation in various domains (see Chen et al. [2016]), including discrete ones (see Zoph et al. [2016]). In particular, Vinyals et al. [2015b] revisited the travelling salesman problem (TSP) with the Pointer Networks, where a recurrent network with non-parametric softmaxes is trained in a supervised manner to predict the sequence of visited cities.
As discussed in Section (10.1), supervised learning is about learning a mapping from training inputs to outputs. Thus, in order to learn an optimisation problem we would need to have access to optimal labels, which is in general not the case. Bello et al, [2017] followed the reinforcement learning (RL) paradigm to tackle combinatorial optimisation, leading to Neural Combinatorial Optimisation. The method allows one to compare the quality of a set of solutions using a verifier (control), and provide some reward feedbacks to a learning algorithm. They considered two approaches based on policy gradients (see Williams et al. [1992]): (1) RL pretraining, uses a training set to optimise a recurrent neural network (RNN) that parameterises a stochastic policy over solutions, using the expected reward as objective. At test time, the policy is fixed, and one performs inference by greedy decoding or sampling. (2) active search, involves no pretraining. It starts from a random policy and iteratively optimises the RNN parameters on a single test instance, again using the expected reward objective, while keeping track of the best solution sampled during the search. The authors found that combining RL pretraining and active search works best in practice. They compared their approach against the supervised learning approach to the TSP proposed by Vinyals et al. [2015b]. The latter consider Pointer Networks which is a neural architecture to learn the conditional probability of an output sequence with elements that are discrete tokens corresponding to positions in an input sequence.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

21.2.2.2 The problem

For simplicity of exposition we are going to present a 2D Euclidean travelling salesman problem (TSP). Given an input graph (see Section (13.1.1.2)), represented as a sequence of n cities in a two dimensional space $s = \{x_i; i = 1, ..., n\}$, where each $x_i \in \mathbb{R}^2$, our objective is to find a permutation of the points π (called a tour) that visits each city once and has the minimum total length. The length of a tour, defined by a permutation π, is given by

$$L(\pi|s) = \|x_{\pi(n)} - x_{\pi(1)}\|_2 + \sum_{i=1}^{n-1} \|x_{\pi(i)} - x_{\pi(i+1)}\|_2$$

where $\|\cdot\|_2$ is the L_2 norm. We want to learn the parameters of a stochastic policy $p(\pi|s)$ [2], which given an input set of points s, assigns high probabilities to short tours and low probabilities to long tours.
Sutskever et al. [2014] proposed a neural network architecture that uses the chain rule to factorise the probability of a tour as

$$p(\pi|s) = \prod_{i=1}^{n} p\big(\pi(i)|\, \pi(< i), s\big)$$

and then uses individual softmax modules to represent each term on the RHS of the above equation. One could then use a sequence to sequence model (seq2seq) to address the TSP where the output vocabulary is $\{1, 2, ..., n\}$. However, such trained network can not generalise to input with more than n cities and we need access to the true output permutation to train the model.
Vinyals et al. [2015b] proposed the Pointer Network that generalises the approach beyond a pre-specified graph size by making use of a set of non-parametric softmax modules, similar to the attention mechanism from Bahdanau et al. [2015]. It allows the model to effectively point to a specific position in the input sequence rather than predicting an index value from a fixed-size vocabulary.
The authors proposed training a pointer network using a supervised loss function comprising conditional log-likelihood, which factors into a cross entropy objective between the network's output probabilities and the targets provided by a TSP solver. However, as pointed out by Bello et al. [2017], the performance of the model is tied to the quality of the supervised labels, getting high-quality labelled data is expensive and may be infeasible for new problem statements. At last, it will be equivalent to replicating the results of another algorithm (TSP solver).

21.2.3 The REINFORCE algorithm

As discussed in Section (17.1.4), the REINFORCE algorithm is a policy gradient method that uses the likelihood ratio trick. Basically it increases the probability that a policy will make a rewarding action given a state s. It is a family of reinforcement learning methods which directly update the policy weights θ through the following rule:

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) v_t \tag{21.2.2}$$

where α is the learning rate, $\pi_\theta(\cdot|s_t)$ is the policy (mapping actions to probabilities) (see Definition (15.3.2)), and v_t is a sample of the value function at time t collected from experience.
It iteratively updates the agent's parameters by computing the policy gradient as follows:

- Update the parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return v_t as an unbiased sample of $Q_{\pi_\theta}(s_t, a_t)$ solve Equation (21.2.2)

[2] The authors use a different notation than ours. It corresponds to the parametric policy $\pi_\theta(s, a)$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Previously to the REINFORCE algorithm, one had to learn a value function through function approximation and then derive a corresponding policy (see Section (17.1)), but often learning a value function can be intractable. This method provides a way to directly optimise policies to get around this problem. See details in Algorithm (66).

Algorithm 66 REINFORCE Algorithm

Require: : training set S, number of training steps T

Initialise θ
for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
  for $t = 1$ to $T - 1$ **do**
    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) v_t$
  end for
end for
Return θ

21.2.4 Applying reinforcement learning

In the case of combinatorial optimisation, Bello et al. [2017] considered reinforcement learning (RL) because these problems have simple reward mechanism that can even be used at test time. They considered two approaches based on policy gradients (see Section (17.1.4)) where we use neural networks to learn the policy. They used the pointer network architecture as the policy model to parametrise the policy $p(\pi|s)$. Their pointer network comprises two recurrent neural network (RNN) modules, encoder and decoder, both of which consist of Long Short-Term Memory (LSTM) cells (see Section (6.2)).

1. The encoder network reads the input sequence s one city at a time, and transforms it into a sequence of latent memory states $\{enc_i\}_{i=1}^n$ where $enc_i \in \mathbb{R}^2$. The input to the encoder network at time step i is a d-dimensional embedding of a 2D point x_i, which is obtained via a linear transformation of x_i shared across all input steps.

2. The decoder network also maintains its latent memory states $\{dec_i\}_{i=1}^n$ where $dec_i \in \mathbb{R}^2$. At each step i, it uses a pointing mechanism to produce a distribution over the next city to visit in the tour. Once the next city is selected, it is passed as the input to the next decoder step. The input of the first decoder step (denoted $< g >$) is a d-dimensional vector treated as a trainable parameter of the neural network.

The attention function takes as input a query vector $q = dec_i \in \mathbb{R}^d$ and a set of reference vectors $ref = \{enc_1, ..., enc_k\}$ where $enc_i \in \mathbb{R}^d$, and predicts a distribution $A(ref, q)$ over the set of k references. This probability distribution represents the degree to which the model is pointing to reference r_i upon seeing query q.
The authors used model-free policy-based Reinforcement Learning to optimise the parameters of the pointer network denoted θ. The training objective is the expected tour length which, given an input graph s, is defined as

$$J(\theta|s) = E_{\pi \sim p_\theta(\cdot|s)}[L(\pi|s)]$$

where the length of a tour $L(\pi|s)$ acts as the reward.
During training, the graphs are drawn from a distribution $\mathcal{S}$ and the total training objective involves sampling from the distribution of graphs

$$J(\theta) = E_{s \sim S}[J(\theta|s)]$$

The parameters are optimised with the policy gradient methods (see Section (17.1.4)) and the stochastic gradient descent (see Section (18.3)). The gradient of the above objective is formulated with the REINFORCE algorithm (see Section (17.1.4.2)) as follows

$$\nabla_\theta J(\theta|s) = E_{\pi \sim p_\theta(\cdot|s)}[\big(L(\pi|s) - b(s)\big)\nabla_\theta \log p_\theta(\pi|s)]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $b(s)$ denotes a baseline function, independent from π, that estimates the expected tour length to reduce the variance of the gradients. Drawing B i.i.d. sample graphs $s_1, s_2, ..., s_B \sim \mathcal{S}$ and sampling a single tour per graph, $\pi_i \sim p_\theta(\cdot|s_i)$, the above gradient is approximated with with Monte Carlo sampling as

$$\nabla_\theta J(\theta) \approx \frac{1}{B}\sum_{i=1}^{B}\Big(L(\pi_i|s_i) - b(s_i)\Big)\nabla_\theta \log p_\theta(\pi_i|s_i)$$

In general it is taken to be an exponential moving average of the rewards obtained by the network over time to account for the fact that the policy improves with training. However, we can not differentiate between the different input graphs. Since a parametric baseline to estimate the expected tour length improves the learning, the authors introduced an auxiliary network, called a critic, and parametrised it by θ_v to learn the expected tour length (see Section (17.1.5)). It is trained with stochastic gradient descent on a mean squared error objective between its prediction $b_{\theta_v}(s)$ and the actual tour lengths sampled by the most recent policy. The objective is given by

$$\mathcal{L}(\theta_v) = \frac{1}{B}\sum_{i=1}^{B}\|b_{\theta_v}(s_i) - L(\pi_i|s_i)\|_2^2$$

where B is the batch size. The critic to map a sequence s into a baseline $b_{\theta_v}(s)$ is made of three neural network modules

1. an LSTM encoder
2. an LSTM process block
3. a 2-layer ReLU neural network decoder

The training algorithm, closely related to the asynchronous advantage actor-critic (A3C) (see Mnih et al. [2016]), is given in Algorithm (67).

Algorithm 67 Actor-critic Training Algorithm

Require: : training set S, number of training steps T, batch size B

```
Initialise pointer network parameters θ
Initialise critic network parameters θ_v
for t = 1 to T do
    s_i ~ SampleInput(S) for i ∈ {1, ..., B}
    π_i ~ SampleSolution(p_θ(·|s_i)) for i ∈ {1, ..., B}
    b_i ← b_{θ_v}(s_i) for i ∈ {1, ..., B}
    g_θ ← (1/B) Σ_{i=1}^{B} (L(π|s) − b(s)) ∇_θ log p_θ(π|s)
    L_v ← (1/B) Σ_{i=1}^{B} ‖b_i − L(π_i)‖_2^2
    θ ← ADAM(θ, g_θ)
    θ_v ← ADAM(θ_v, ∇_θ L_v)
end for
Return θ
```

Since the evaluation of a tour length is inexpensive, one can follow classical search by considering multiple candidate solutions per graph and selecting the best one. The authors considered two search strategies

1. Sampling: sample multiple candidate tours from the stochastic policy $p_\theta(\cdot|s)$ and selecting the shortest one.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. Active search (on untrained model): the idea is to reward information obtained from the sampled solutions by refining the parameters of the stochastic policy p_θ during inference to minimise $J(\theta|s)$ on a single test input s. It applies policy gradients as in Algorithm (67), but it draws Monte Carlo samples over candidate solutions $\pi_1, ..., \pi_B \sim p_\theta(\cdot|s)$ for a single test input. It also consider an exponential moving average baseline. Note, it is independent of distribution.

When it comes to solving other optimisation problems, one should use Pointer Network when the output is a permutation or a subset of the input, and the seq2seq model for other kinds of structured outputs. When feasibility is known at decoding time one simply assign a zero probability to a non-feasible solution (branches). However, for many combinatorial problems finding feasible solutions is a challenge as one need to search subtrees of branches. Rather than explicitly constraining the model to only sample feasible solutions, one can let the model learn the constraints by augmenting the objective function with a penalising term, similarly to penalty methods in constrained optimisation (see Section (18.7)). However it does not guarantee to always sample feasible solutions at inference time.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Part VI

Applications

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 22

Mathematical Finance

Since financial markets exhibit non-Gaussian response time distribution leading to more skewness and kurtosis than it is consistent with the Geometric Brownian Motion model of Black-Scholes, neural networks (NNs) are good candidates to learn these highly nonlinear relationships. We saw in Chapter (6) that recurrent neural networks could be used to predict the stock market. Further, we saw in Chapter (10) that supervised learning was concerned with solving function estimation problems. These methods can be used in finance to calibrate option prices in view of interpolating / extrapolating the volatility surface. It can also be used in risk management to fit options prices at the portfolio level in view of performing some credit risk analysis. Further, we saw in Chapter (17) that reinforcement learning (RL) consisted in solving problems involving an agent interacting with an environment, which provides numeric reward signals. The goal being to learn how to take actions in order to maximise reward. As such it can be used for selecting optimal portfolios. We will see in this chapter that it can also be used to price European and American options when the principle of hedging first and pricing second is applied. Contrary to classical option pricing theory which is rooted in stochastic calculus and continuous-time models, machine learning is rooted in dynamic programming (DP) and reinforcement learning. However, solving financial problems in that setting requires an exhaustive amount of data, which is in general not there. As such, one must rely on some models to generate the missing data.

22.1 Option pricing

22.1.1 The framework

22.1.1.1 Some definitions

22.1.1.1.1 Asset returns Return series are easier to handle than price series due to their more attractive statistical properties and to the fact that they represent a complete and scale-free summary of the investment opportunity (see Campbell et al. [1997]). Expected returns need to be viewed over some time horizon, in some base currency, and using one of many possible averaging and compounded methods.
Holding the asset for one period of time, from date t to date $(t+1)$, would result in a simple gross return

$$1 + R_{t,t+1} = \frac{S_{t+1}}{S_t} \tag{22.1.1}$$

where the corresponding one-period simple net return, or simple return, $R_{t,t+1}$, is given by

$$R_{t,t+1} = \frac{S_{t+1}}{S_t} - 1 = \frac{S_{t+1} - S_t}{S_t}$$

More generally, we let

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$R_{t-d,t} = \frac{\nabla_d S_t + D_{t-d,t}}{S_{t-d}}$$

be the discrete return of the underlying process where $\nabla_d S_t = S_t - S_{t-d}$ with period d and where $D_{t-d,t}$ is the dividend over the period $[t-d, t]$. For simplicity we will only consider dividend-adjusted prices with discrete dividend-adjusted returns $R_{t-d,t} = \frac{\nabla_d S_t}{S_{t-d}}$. Hence, holding the asset for d periods, between the dates $t-d$ and t, gives a d-period simple gross return

$$\begin{aligned} 1 + R_{t-d,t} = \frac{S_t}{S_{t-d}} &= \frac{S_t}{S_{t-1}} \times \frac{S_{t-1}}{S_{t-2}} \times \cdots \times \frac{S_{t-d+1}}{S_{t-d}} \\ &= (1 + R_{t-1,t})(1 + R_{t-2,t-1}) \cdots (1 + R_{t-d,t-d+1}) = \prod_{j=1}^{d}(1 + R_{t-j,t-j+1}) \end{aligned} \tag{22.1.2}$$

so that the d-period simple gross return is just the product of the d one-period simple gross returns, which is called a compound return. In that setting, the price of the underlying asset becomes

$$S_t = S_{t-d} \prod_{j=1}^{d}(1 + R_{t-j,t-j+1})$$

In the financial industry, most measures of returns and indices use change of returns $R_t = R_{t-1,t}$ defined as $\frac{S_t - S_{t-1}}{S_{t-1}}$ where S_t is the price of a series at time t. However, some investors and researchers prefer to use returns based on logarithms of prices $r_t = \log \frac{S_t}{S_{t-1}}$ or compound returns.

22.1.1.1.2 Derivatives A derivative, or contingent claim, is a financial instrument whose value depends on the values of other more basic underlying variables being the prices of traded securities. An option is a security giving the right to buy or sell an asset, subject to certain conditions, within a specified period of time. The holder of an option has the right to exercise the option but it is not an obligation. European, or vanilla, options such as a call option give the holder the right to buy the underlying asset by a certain date called the maturity for a specified price called the exercise price or strike price, while the put option gives him the right to sell it. We let K be the strike price and S_T be the underlying price at time T, then the payoff from a long position in a European call option is

$$\max\big(S_T - K, 0\big) = (S_T - K)^+$$

so that the option will be exercised if $S_T > K$.
We display in Figure (22.1) the call price and its linear approximation versus different strike levels.
As a result of the concept of absence of arbitrage opportunities (AAO) (see Harrison and Kreps [1979]), contingent claims can be valued by taking expectation of their discounted payoffs under the risk-neutral measure. Therefore, the price of a European contingent claim $C(t, x)$ on $]0, T] \times]0, +\infty[$ under the risk-neutral measure $\mathbb{Q}$ is

$$C(t, X_t) = E^Q[e^{-\int_t^T r_s ds} h(X_T)|\mathcal{F}_t] \tag{22.1.3}$$

where X_T is a $\mathbb{Q}$-random variable and h is a sufficiently smooth payoff function.

22.1.1.2 Defining the replicating portfolio

Initial work on the valuation of options has been expressed in terms of warrants (see Sprenkle [1961], Samuelson [1965] and Samuelson et al. [1969]), all providing valuation formulas involving one or more arbitrary parameters. Thorp et al. [1967] obtained an empirical valuation formula for warrants which they used to calculate the ratio of shares of stock to options needed to create a hedge position by going long in one security and short in the other.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 22.1: The call price and its linear approximation vs strikes.

However, they failed to see that in equilibrium, the expected return on such a hedge position must be equal to the return on a riskless asset. Using this equilibrium condition, Black et al. [1973] derived a theoretical valuation formula. To do so, they assumed the following ideal conditions in the market

1. The short-term interest rate is known and constant through time.
2. The stock price follows a random walk in continuous time with a variance rate proportional to the square of the stock price.
3. The stock pays no dividends or other distributions.
4. The option is European.
5. There are no transaction costs in buying or selling the stock or the option.
6. It is possible to borrow any fraction of the price of a security to buy it or to hold it, at the short-term interest rate.
7. There are no penalties to short selling.

Under these assumptions, the value of the option will only depend on the price of the stock and time, so that it is possible to create a hedged position, consisting of a long position in the stock and a short position in the option. In

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

a complete market, every terminal payoff can be replicated by a portfolio, so that the problem of defining the price of an option is linked with that of finding a replicating portfolio. That is, using the option premium, one can buy and sell the underlying to hedge risk against the derivative. Following that argument, Black and Scholes defined the price of a derivative option as the price of its hedge. It leads to the formulation of the local no-arbitrage condition (see Jamshidian [1991]):

Definition 22.1.1 *The local no-arbitrage condition*
The local no-arbitrage condition states that in a one-factor case, a portfolio can be constructed from any two securities and continually rebalanced at each time, in such a way that it is instantaneously hedged, in the sense that its instantaneous return is certain and hence must equal the short-term, spot interest rate.

Thus, knowing the dynamics of the underlying price we need to formalise the evolution of the self-financing replicating portfolio dynamically managed (without adding or removing capital during the trading period). Following the approach in Section (28.2.4.1), we put ourself in the Black-Scholes world and we assume a market made of a risk-free asset S_t^0 and a risky asset with price S_t. We model the dynamics of the future stock prices under the historical measure $\mathbb{P}$ as

$$\begin{aligned} dS_t &= S_t\big(\mu_t dt + \sigma_t d\widehat{W}_t\big) \\ \mu_t &= r + \sigma_t \lambda_t \end{aligned}$$

We let S_t^0 be the cash describing the value of \$1 daily compounded at the bank. It is a risk-free asset earning rdt in the time interval $[t, t+dt]$ known at time t. Its dynamics are given by

$$dS_t^0 = S_t^0 r dt \ , \ S_0^0 = 1$$

Integrating in the range $[0, t]$, we get the money market $S_t^0 = e^{rt}$. We consider the discounted price $\overline{S}_t = \frac{S_t}{S_t^0}$ to be a martingale with dynamics given by

$$d\overline{S}_t = \overline{S}_t\big((\mu_t - r)dt + \sigma_t d\widehat{W}_t\big)$$

We consider the process L_t satisfying

$$dL_t = -\frac{(\mu_t - r)}{\sigma_t} L_t d\widehat{W}_t \ , \ L_0 = 1$$

Using Girsanov's theorem, the process $\overline{S}_t$ satisfies

$$d\overline{S}_t = \overline{S}_t \sigma_t dW_t^*$$

where $W_t^* = W_t + \int_0^t \frac{(\mu_s - r)}{\sigma_s} ds$ is a Q-Brownian motion. The probability Q is equivalent to to P.
We denote the quantity of cash that one can hold as $\beta(t)S_t^0$. We build a self-financing portfolio V made of cash and stocks with value at time t being

$$V_t = \alpha(t)S_t + \beta(t)S_t^0$$

and we denote the value invested in stocks as $\pi_t = \alpha(t)S_t$. Over a very short period of time the variations of the portfolio are only due to the variations of the asset price and the interest rate rewarding the cash held at the bank. Hence, the portfolio dynamics are given by

$$dV_t = \alpha(t)dS_t + \beta(t)dS_t^0 \tag{22.1.4}$$

Replacing the respective dynamics in the above equation, the self-financing portfolio satisfies the stochastic differential equation (SDE)

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$dV_t = r_t V_t dt + \pi_t \sigma_t (d\widehat{W}_t + \lambda_t dt)$$

Market completeness is equivalent to assuming that markets are efficient, that is, the price of an asset at a given time contains all past information as well as the market expectation of its future value. This means that there exists a strategy leading to a final P/L of null value in all possible future configuration. As a result, we can conclude that the option price is equal to the value of the replicating portfolio at the initial time. A direct application of the above assumption is that we can express the Black-Scholes price dynamics as

$$dC_{BS}(t, S_t) = r_t C_{BS}(t, S_t)dt + \pi_t \sigma_t (d\widehat{W}_t + \lambda_t dt)$$

Using Girsanov's theorem, we can re-express that price dynamics under the risk-neutral measure $\mathbb{Q}$ by applying the change of measure

$$W_t = \widehat{W}_t + \int_0^t \lambda_s ds$$

where W_t is a $\mathbb{Q}$-Brownian motion. Therefore, the Black-Scholes price at all future time is a solution to the following stochastic differential equation with terminal condition

$$\begin{aligned} dC_{BS}(t, S_t) &= r_t C_{BS}(t, S_t)dt + \pi_t \sigma_t dW_t \\ C_{BS}(T, S_T) &= h(S_T) \end{aligned} \tag{22.1.5}$$

where the pair $(\alpha(t), \beta(t))$ allows replication of the contingent claim paying $h(S_T)$ at maturity.
Note, the local volatility of the call option is more volatile than the volatility of the underlying asset. It is given by

$$\sigma_{BS}(t) = \frac{\alpha(t) S_t \sigma_t}{C_{BS}(t, S_t)} \geqslant \sigma_t \tag{22.1.6}$$

Note, a consequence of the option price being expressed under the risk-neutral measure is that it neither depend on the return μ of the risky asset nor the market price of risk λ. However, the risk due to the variations of the risky asset is still present and significantly impact the option price via the volatility parameter σ. Thus, to avoid arbitrage, a market price of risk is computed such that the discounted spot price is a martingale with respect to the equivalent probability measure. In the risk-neutral measure, the drift of the spot price is equal to the risk-free rate so that the key parameter becomes the volatility of the dynamics. Volatility is a key parameter when risk managing option prices as it allows the portfolio manager, or trader, to define his hedging strategies by computing the necessary Greeks. The Vega is a risk measure quantifying exposure to a misspecified volatility.

22.1.1.3 The BS-formula

In the special case where we assume that the stock prices $(S_t^x)_{t \geqslant 0}$ are lognormally distributed and pay a continuous dividend, we can derive most of the contract market values in closed-form solution. Balck-Scholes [1973] formalised a method to infer such a solution by developing a theory of hedging and replication by dynamic strategies. They derived the price of a call option seen at time t with strike K and maturity T as

$$C_{BS}(t, x, K, T) = xe^{-q(T-t)} N(d_1(T-t, F(t,T), K)) - Ke^{-r(T-t)} N(d_2(T-t, F(t,T), K)) \tag{22.1.7}$$

where $F(t,T) = xe^{(r-q)(T-t)}$ is the forward price and $N(\cdot)$ is the normal cumulative distribution function (CDF). Further, we have

$$d_2(t, x, y) = \frac{1}{\sigma\sqrt{t}} \log \frac{x}{y} - \frac{1}{2}\sigma\sqrt{t} \text{ and } d_1(t, x, y) = d_2(t, x, y) + \sigma\sqrt{t}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We describe a few Greeks in the Black-Scholes formula that will be used later on to devise our parametric model. The option in the Black-Scholes model is hedged with a portfolio containing

$$\Delta(t, S_t) = \partial_x C_{BS}(t, S_t, K, T) = Re(t, T)N(d_1(T - t, S_t e^{(r-q)(T-t)}, K)) \tag{22.1.8}$$

stocks, since $\frac{d}{dx}d_i = \frac{1}{x\sigma\sqrt{T-t}}$ for $i = 1, 2$. Note, $x = S_t$ and $Re(t, T) = e^{-q(T-t)}$. We let $P(t, T)$ be the discount factor, $Re(t, T)$ be the repo factor and we define

$$\eta = \frac{K}{F(t, T)} = \frac{KP(t, T)}{xRe(t, T)} \tag{22.1.9}$$

to be the forward moneyness of the option. Expressing the strike in terms of the forward price $K = \eta F(t, T)$, the call price in Equation (22.1.7) becomes

$$C_{TV}(t, x, K, T)\big|_{K=\eta F(t,T)} = xe^{-q(T-t)}\big(N(d_1(\eta, \omega(T - t)) - \eta N(d_2(\eta, \omega(T - t)))\big) \tag{22.1.10}$$

where $d_2(\eta, \omega(t)) = \frac{1}{\sqrt{\omega(t)}} \log \frac{1}{\eta} - \frac{1}{2}\sqrt{\omega(t)}$, or equivalently

$$d_2(\eta, \omega(t)) = -\frac{1}{\sqrt{\omega(t)}} \log \eta - \frac{1}{2}\sqrt{\omega(t)} \text{ and } d_1(\eta, \omega(t)) = d_2(\eta, \omega(t)) + \sqrt{\omega(t)} \tag{22.1.11}$$

which only depends on the forward moneyness and the total variance.

22.1.1.4 The implied volatility

In the BS-model, the volatility is a parameter quantifying the risk associated to the returns of the underlying asset, and it is the only unknown variable. In principle, one should use the future volatility in the BS-formula, but its value is not known and needs to be estimated. As a result, practitioners use the implied volatility (IV) when managing their books, assuming that the IV bears valuable information on the asset price process and its dynamics.
We consider non-negative price processes, and assume a perfectly liquid market for European calls. That is, call option prices for all strikes $K > 0$, and maturities $T \geqslant 0$ are known in the market. We define a price surface as follows (see Roper [2010]):

Definition 22.1.2 *A call price surface parametrised by s is a function*

$$\begin{aligned} C &: [0, \infty) \times [0, \infty) \to \mathbb{R} \\ (K, T) &\to C(K, T) \end{aligned}$$

along with a real number $s > 0$.

However, market practice is to use the implied volatility when calculating the Greeks of European options. Thus, we let the implied volatility (IV) be a mapping from time, spot prices, strike prices and expiry days to $\mathbb{R}^+$

$$\Sigma : (t, S_t, K, T) \to \Sigma(t, S_t; K, T)$$

Hence, we define the implied volatility as follow:

Definition 22.1.3 *Given the option price $C(t, S_t, K, T)$ at time t for a strike K and a maturity T, the market implied volatility $\Sigma(t, S_t; K, T)$ satisfies*

$$C(t, S_t, K, T) = C_{BS}(t, S_t, K, T; \Sigma(K, T)) \tag{22.1.12}$$

where $C_{BS}(t, S_t, K, T; \sigma)$ is the Black-Scholes formula for a call option with volatility σ.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Given (S, K, T) and the price of an option, there is a unique implied volatility associated to that price, since

$$\partial_\sigma C_{BS}(t, S_t, K, T; \sigma) > 0$$

Thus, the implied volatility is obtained by inverting the Black-Scholes formula $C_{BS}^{-1}(C(t, S_t, K, T); K, T)$. Consequently, we refer to the two-dimensional map

$$(K, T) \to \Sigma(K, T)$$

as the implied volatility surface. Note, we will some time denote $\Sigma_{BS}(K, T)$ the Black-Scholes implied volatility.

Definition 22.1.4 *IV map*
We let $\mathcal{I}$ be the pricing input space. The function $\varphi : \mathcal{I} \to \mathbb{R}_+$ given by

$$(t, S_t; K, T) \to \Sigma_{BS}(K, T)$$

is the implied volatility map.

A representation of the volatility surface is given in Figure (22.2).

Figure 22.2: EuroStock Volatility Surface on the $8/12/2003$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

When visualising the IVS, it makes more sense to consider the two-dimensional map $(\eta, T) \to \Sigma(\eta, T)$ [1] where η is the forward moneyness (corresponding to the strike $K = \eta F(t, T)$). We can also use the forward log-moneyness $\overline{\eta}$ such that the two-dimensional map becomes $(\overline{\eta}, T) \to \Sigma(\overline{\eta}, T)$ [2]. Further, we let the total variance be given by

$$\omega(\eta, T) = \nu^2(\eta, T) = \Sigma^2(\eta, T)(T - t)$$

and let the implied total variance satisfies

$$C(t, S_t, K, T) = C_{TV}(t, S_t, \eta F(t, T), T; \omega(\eta, T)) \tag{22.1.13}$$

where the two-dimensional map $(\eta, T) \to \omega(\eta, T)$ is the total variance surface. Note, $\sqrt{\omega(\overline{\eta}, T)}$ corresponds to the time-scaled implied volatility in forward log-moneyness form.

22.1.1.5 The no-arbitrage conditions

The shape of the implied volatility surface is constrained by absence of arbitrage (AOA) which are as follow:

- we must have $\Sigma(K, T) \geqslant 0$ for all strikes K and maturity T.
- at any given maturity T, the skew can not be too steep (to avoid butterfly arbitrage).
- the term structure of implied volatility can not be too inverted (to avoid calendar spread arbitrage).

Further, understanding the behaviour of the implied volatility (IV) surface for far expiries and far strikes is fundamental for extrapolation problems. Thus, various necessary conditions for an implied volatility surface to be properly defined have been proposed (see Durrleman [2003], Lee [2005]). A summary can be found in textbook by Fengler [2005]. Later, Roper [2010] established necessary and sufficient conditions for an IVS to be free from static arbitrage. These conditions were expressed in terms of implied volatility providing a complete characterisation of an IVS free from static arbitrage (see Rogers et al. [2010]).

Definition 22.1.5 *Arbitrage-Free Implied Volatility Surface*
Assuming deterministic interest and repo rates, an implied volatility surface is free from static arbitrage if and only if the following conditions are satisfied

(A1) (Monotonicity of total variance)
$\forall x$ *and* $T > 0$, $\partial_T \omega(x, T) \geqslant 0$

(A2) (Convexity in K)
$\forall T > 0$, $\partial_{KK} C(K, T) \geqslant 0$

(A3) (Large-Moneyness Behaviour)
$\lim_{\overline{\eta} \to \infty} d_1(\overline{\eta}, \omega(\overline{\eta}, t)) = -\infty$ *and* $\lim_{\overline{\eta} \to \infty} d_2(\overline{\eta}, \omega(\overline{\eta})) = -\infty$

(A4) (Small-Moneyness Behaviour)
$\lim_{\overline{\eta} \to -\infty} d_2(\overline{\eta}, \omega(\overline{\eta}, t)) = \infty$ *and* $\lim_{\overline{\eta} \to -\infty} d_1(\overline{\eta}, \omega(\overline{\eta})) = \infty$

[1] Here $\Sigma(\eta, T)$ must be understood as $\Theta(\eta, T) \to \Sigma(F(t, T)\eta, T)$
[2] Here $\Sigma(\overline{\eta}, T)$ must be understood as $\Theta(\overline{\eta}, T) \to \Sigma(F(t, T)e^{\overline{\eta}}, T)$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

22.1.1.6 Characterising the smile dynamics

22.1.1.6.1 The regimes of volatility We present some recently proposed deterministic implied volatility models, defined by assuming that either the per-delta or the per-strike implied volatility surface has a deterministic evolution. We then give some rules of thumb used by practitioners to compute their plain-vanilla prices. To do so, we look at some standard evolutions of the implied volatility surface denoted by $\Sigma(t, S_t; K, T)$ and first described by Derman [1999] and extended by various authors such as Daglish et al. [2006]. It provides an indication of some possible behaviour for the smile that one might expect, even though they are not all **arbitrage-free**.

1. sticky delta : the strike of the option is rescaled according to how the current spot evolved with respect to the spot at inception
   $\Sigma(t, S_t; K, T) = \Sigma_0^{obs}(K\frac{S_0}{S_t}, T - t)$
2. absolute floating : the future implied volatility is obtained from the original smile surface by simply reducing time to maturity and linearly offsetting the strike by how much the spot has moved
   $\Sigma(t, S_t; K, T) = \Sigma_0^{obs}(K + (S_0 - S_t), T - t)$
3. absolute sticky (sticky strike) : any dependence on the current spot level or calendar time is ignored
   $\Sigma(t, S_t, K, T) = \Sigma_0^{obs}(K, T - t)$

where Σ^{obs} is the current smile. These regimes of volatility were slightly extended by Derman [2008], who for simplicity of exposition, considered the linear smile (only true when the strike is near the money)

$$\Sigma(t_0, S_0; K, T) = a - b(K - S_0)\ , b > 0$$

where a can be seen as the ATM-volatility, and studied its variation when the underying moves away from S_0.

1. sticky moneyness : The dynamics of option prices derive entirely from the moneyness

   $$\Sigma(t, S_t; K, T) = a - b(\frac{K}{S_t} - 1)S_0\ , b > 0$$

   It shifts the skew as the stock price moves by adjusting for moneyness, assuming that the market mean reverts to the value a independently from market level. When the skew is nearly linear we get $\Sigma \approx \Sigma(S_t - K)$, so that the implied volatility rises when S rises. In the BS-delta (see Equation (22.1.8)), the dependence on K and S occurs only through the moneyness η, such that for ATM option we get $|\Delta| = \frac{1}{2}$. However, the delta depends on the scaled moneyness $\frac{1}{\sqrt{\omega(t)}}\log\eta$, which should be the fixed point. In presence of negative skew, the IV increases with increasing spot, leading to a greater delta than BS-delta. Stochastic volatility models and jump-diffusion models have this property (assuming the other variables do not change).
2. sticky implied tree : In the implied tree, local volatility increases as the underlying decreases. Assuming that the IV is approximately the linear average of local volatilities between the initial spot price and the strike price, we can extract the future local and implied volatilities from current implied volatilities. We get the linear approximation to the skew

   $$\Sigma(t, S_t; K, T) = a - b(K + S_t - 2S_0)\ , b > 0$$

   so that the implied volatilities decrease as K or S increases, leading to a smaller delta than BS-delta. Hence, when linearly approximating the local volatility we get $\Sigma \approx f(K + S_t)$ and volatilities are inversly correlated with the underlying spot price. The sticky implied tree regime corresponds to a fear of higher market volatility in the case of a fall of the spot price. Local volatility models have this property.

However, these simplified regimes of volatility have a couple of major drawbacks. First of all, they are not consistent with market observable data, and secondly the way they are used to price forward call options is not derived from the theory of No Arbitrage.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

22.1.1.6.2 Space homogeneity versus space inhomogeneity Even though the dynamics of implied volatility surface (IVS) are neither stationary nor Markovian, we saw in Section (22.1.1.6.1) that they could be classified in terms of regimes which might change over time. These regimes explain the evolution of the vanilla smiles in terms of the strike price and the underlying spot price. At the two extremes, the smile can either be expressed in terms of moneyness, or in terms of absolute value for the strike and spot prices. The former is generated by space homogeneous models, like the Heston model, and follow the sticky delta rule. The latter is generated by space inhomogeneous models, such as certain types of local volatility models, and follow the sticky implied tree rule. In between, practitioners developed a range of universal models, where the diffusion term is a mix of stochastic volatility and local volatility (see Lipton [2002], Blacher [2001]). The sticky delta rule leads to a delta greater than the BS-delta, meaning that the dynamics of the IV follow that of the underlying asset. In the sticky implied tree rule the delta is smaller than the BS-delta, meaning that the dynamics of the IVS is inversly related to the underlying asset. It is interesting to note that some authors justified their models by the type of dynamics they produce. For Blacher [2001], homogenous smile do not predict a systematic move of the relative smile when spot changes. Observing the equity index, the smile does not move coincidentally with the underlying asset, so that to modify the dynamics of the smile he had to re-introduce inhomogeneity in the stochastic model. On the contrary, for Hagan et al. [2002] local volatility models (LVMs) predict smile dynamics opposite of observed market behaviour. LVMs predict that the smile shifts to higer prices when the underlying asset decreases, and that the smile shifts to lower price when spot price increases. Observing interest rate smiles, where asset prices and market smiles move in the same direction, they devised the SABR model, which is space inomogeneous for $\beta \neq 1$ and allows to capture the correct dynamics of the smile (in the rates market). Thus, we see that the specific model chosen characterise a particular type of dynamics for the IVS, which in turn impose a particular delta hedging ratio.

22.1.1.7 Pricing formulation

We saw in Section (22.1.1.3) that when the stock price is log-normally distributed with constant model parameters, the European call prices satisfy $C(t,x) = C_{BS}(t,x,K,T)$ for $t \in [0,T]$. We also know that when the model parameters are general Markov processes we can no-longer use the Black-Scholes Equation (22.1.7). Assuming constant spot rate and repo rate, the pricing formula for a call option with maturity T is given by

$$E^Q\left[C_{BS}(t,x;I(t,T))|\mathcal{F}_t\right], I(t,T) = \sqrt{\frac{1}{T-t}\int_t^T \sigma_s^2 ds}$$

In the special case of uncorrelated stochastic volatility, Hull and White [1987] obtained the pricing formula for a call option, and derived, in the case of correlated volatility, an approximation from a second-order Taylor series expansion around a constant volatility specification. Closed-form solutions with correlated stochastic volatility were found (see Heston [1993]) but not expressed in terms of the Black-Scholes formula. However, we still want to express the solution to the European call options in terms of the Black-Scholes formula, which can then be seen as a function of the stochastic spot rate, repo rate and volatility.
More recently, some authors decomposed the derivative price with general volatility process into combination of Black-Scholes terms by means of the Ito's lemma (see Alos [2004], Alos et al. [2007]). Bloch [2010a] have described a general pricing approximation technique for call options in a jump-diffusion model with stochastic interest rates and stochastic repo rates. At maturity, the European option prices must satisfy

$$C(T,x) = C_{BS}\Big(t,x,K,T;\sqrt{\frac{1}{T-t}\int_t^T \sigma_s^2 ds},\sqrt{\frac{1}{T-t}\int_t^T r_s ds},\sqrt{\frac{1}{T-t}\int_t^T q_s^2 ds}\Big)\Big|_{t=T}$$

and since $e^{-\int_0^t r_s ds}C(t,x)$ is a $\mathbb{Q}$-martingale, we get

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$e^{-\int_0^t r_s ds} C(t,x) = E^Q[e^{-\int_0^T r_s ds} C(T,x)|\mathcal{F}_t]$$
$$= E^Q[e^{-\int_0^T r_s ds} C_{BS}\Big(t,x,K,T;\sqrt{\frac{1}{T-t}\int_t^T \sigma_s^2 ds},\sqrt{\frac{1}{T-t}\int_t^T r_s ds},\sqrt{\frac{1}{T-t}\int_t^T q_s^2 ds}\Big)\Big|_{t=T}|\mathcal{F}_t]$$

22.1.1.8 The intrinsic risk

22.1.1.8.1 Problems with continuous models We saw in Section (22.1.1.2) that the BS-formula was derived from the assumption that market returns were i.i.d. random variables and that the option price could be continuously hedged. That is, the continuous-time limit is assumed so that pricing by replication becomes exact in this limit, and all risks are instantaneously eliminated. Taking the limit, the option becomes risk-less, but also redundant. While this limit makes mathematical sense, it is note the case in practice. Nonetheless, this model became the foundation of mathematical finance.

Continuous-time models developed where stochastic calculus is used to model the infinitesimal movements of the underlying asset. Denoting dX a Wiener process, we use Ito's lemma to understand its behaviour in the limit $dt \to 0$. It relates the small change in a function of a random variable to the small change in the random variable itself. A heuristic approach to Ito's lemma is based on the Taylor series expansion. As a short cut one consider the following result: with probability one, $dX^2 \to dt$ as $dt \to 0$. Some textbooks even display multiplication table (see Table (22.1)).

	dt	$dX_{i,t}$	$dX_{j,t}$
dt	0	0	0
$dX_{i,t}$	0	dt	0
$dX_{j,t}$	0	0	dt

Table 22.1: Multiplication table

However, in practice, prices are quoted at discrete-time intervals Δt. As a result, traders use the BS-formula to compute the market prices, but approximate their delta (the Greeks) to make them discrete. For instance, one uses central difference (or other schemes) with a particular space step ΔX. As such it becomes model dependent (see Benhamou [2001]). That is, we need to assume something about the dynamics of the market prices in the discrete-time interval Δt for the space step ΔX. One solution is to consider the regimes of volatility described in Section (22.1.1.6), but the resulting dynamics are not arbitrage-free. In continuous time models with volatility a deterministic function of time and stock price, such as the Dupire model, the volatility surface is assumed Markovian and stationary, giving a unique solution to the conditional densities and hence to the forward volatility. However, the implied volatility surface is neither stationary nor Markovian, but stochastic.

Another solution is to complexify the dynamics of the underlying asset such as using stochastic volatility model. However, in a large enough discrete-time interval Δt, the space step ΔX may no-longer be modelled with an Ito's process, which describes infinitesimal movements of the spot prices. This is problematic in the case of market shocks, when the market prices exhibit large movements. One solution is to modify the dynamics of the underlying asset by adding jumps, but the resulting prices are no-longer built from the replicating strategies.

22.1.1.8.2 Accounting for risk Since intrinsic risk (non-redundant options) is at the heart of option pricing (see Appendix (28.2.4)), and since a replicating strategy can only be discrete, the pricing formula should be considered in a discrete time setting. The Kolmogorov-Compatibility condition (see Appendix (22.6.4)) states that when the number of fixing dates in a model is finite, there is an infinity of conditional densities. As a result, one has to consider all the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

transition probabilities and their implication to the dynamics of market prices.
Some authors such as Follmer et al. [1989] and Schweizer [1995] proposed some incomplete market models, where, assuming second-order processes, hedging is presented as a quadratic risk minimisation problem. That is, the chosen risk measure corresponds to a mean-variance criterion (see Remark (28.2.7)). Note, this setting does not account for jumps, and the underlying asset is a predictable process and not an optional process (see Appendix (25.1.1.1)).

22.1.1.8.3 Example: the PnL explain All the problems associated with the use of continuous-time models to price and hedge options led market risk professionals to come up with some methods to understand the dichotomy between market prices and the cost of replication of these options. One of this solution is the profit and loss (PnL) explain which consist in decomposing the price of an option with Taylor expansion series [3], but with arbitrary discrete-time intervals Δt and space steps ΔX (see Appendix (22.6.3)). The aim being to approximate the option price with a finite number of terms (Taylor polynomial). This is a heuristic way of measuring the error of discretely hedging an option while pricing it with a continuous model. Obviously, the risk occurs in presence of large movements from the underlying asset. The larger the discrete-time interval Δt, the larger the risk. As such, there is no reason for the PnL explain to be close to zero unless the space steps ΔX are infinitesimal during that period of time. Nonetheless, the experts from market risk do everything in their power to make the PnL zero. That is, while the traders use a continuous model to price and hedge the option, they try to come up with a way of explaining how this option would move in that discrete time-interval given the market space step ΔX (which can be non-negligible).

22.1.1.9 Replicating strategy

We can rewrite the dynamics of the portfolio in Equation (22.1.4) under the risk-neutral measure by applying the change of measure (see Girsanov's theorem)

$$W_t = \widehat{W}_t + \int_0^t \lambda_s ds$$

where W_t is a $\mathbb{Q}$-Brownian motion. We get

$$dV_t = rV_t dt + \pi_t \sigma_S dW_t$$

Given the money market price $M(t) = e^{rt}$, we use the relation

$$M(t)d(M^{-1}(t)V(t)) = dV(t) - rV(t)dt$$

to rewrite the dynamics of the portfolio as

$$d(M^{-1}(t)V(t)) = M^{-1}(t)\pi_t \sigma_S dW_t$$

where $\pi_t = \alpha(t)S_t$. Integrating in the range $[t, T]$, we get

$$V(T) = \frac{M(T)}{M(t)}V(t) + M(T)\int_t^T \alpha(u)\overline{S}(u)\sigma_S dW(u)$$

where $\overline{S}(t) = \frac{S(t)}{M(t)}$ is the discounted price. The latter is a square-integrable martingale with dynamics given by

$$\frac{d\overline{S}(t)}{\overline{S}(t)} = \sigma_S dW(t)$$

Replacing in the above equation, we get

[3] A Taylor series is a representation of a function as an infinite sum of terms that are calculated from the values of the function's derivatives at a single point. The Taylor series requires the knowledge of the function on an arbitrary small neighbourhood of a point. That is, the error is very small in a neighbourhood of the point where it is computed, while it may be very large at a distant point.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$V(T) = \frac{M(T)}{M(t)}V(t) + M(T)\int_t^T \alpha(u)d\overline{S}(u)$$

Since $\overline{S}(t)$ is a square-integrable martingale, so is the discounted portfolio value $\overline{V}(t) = \frac{V(t)}{M(t)}$. Thus, we get

$$\overline{V}(T) = \overline{V}(t) + \int_t^T \alpha(u)d\overline{S}(u)$$

As discussed in Appendix (22.6.1), in the BS-framework the quantity of stock is

$$\alpha(t) = \frac{\partial C_{BS}(t,T)}{\partial S}$$

We let $\{P(t); t \in [0,T]\}$ be a price process satisfying

$$P(T) = h(S(T)) \text{ a. s.}$$

where $h(\cdot)$ is a smooth convex function. In a complete market, the arbitrage price of a European option is given by

$$P_E(t) = M(t)E[\frac{h(S(T))}{M(T)}|\mathcal{F}_t] \, , 0 \leqslant t \leqslant T$$

and the discounted price $\overline{P}_E(t) = \frac{P_E(t)}{M(t)}$ is a martingale.
El Karoui et al. [1998] defined the tracking error associated to the pair (P, α) as the process

$$e(t) = V_\alpha(t) - P(t)$$

with initial condition $V_\alpha(0) = P(0)$. If the discounted tracking error $\overline{e}(t) = \frac{e(t)}{M(t)}$ is

1. identically equal to zero, then (P, α) is a replicating strategy.
2. non-decreasing, then (P, α) is a superstrategy.
3. non-increasing, then (P, α) is a substrategy.

A hedger incorrectly estimating the volatility of the stock will incorrectly compute the option price and the associated hedging portfolio. In a complete market there exists a unique replicating strategy for the European option.
We now consider an agent shorting a European option and hedging it with a replicating portfolio. We assume σ to be the true volatility process, and we let $\{S_\gamma(t); t \in [0,T]\}$ be a stock price with misspecified volatility $\gamma = \gamma(t, S_\gamma(t))$, which is a function of time and current stock price. The dynamics of the stock price with the volatility γ are given by

$$\frac{dS_\gamma(t)}{S_\gamma(t)} = rdt + \gamma dW(t) \, , S_\gamma(t) = x \, , x > 0$$

In that framework, the value of the European option is denoted by $C_\gamma(t, S(t))$. Observing the true stock price $S(t)$, the hedger computes the incorrect price $P_\gamma(t) = C_\gamma(t, S(t))$ and uses the hedging portfolio $\alpha_\gamma(t) = \partial_x C_\gamma(t, S(t))$. Starting with initial value $C_\gamma(0, x)$, his replicating portfolio evolves according to Equation (22.1.4). In that setting, the tracking error $e_\gamma(t) = V_{\alpha_\gamma}(t) - P_\gamma(t)$ is given by

$$e_\gamma(t) = \frac{1}{2}M(t)\int_0^t M^{-1}(u)\big(\gamma^2 - \sigma^2\big)S^2(u)\frac{\partial^2}{\partial x^2}C_\gamma(u, S(u))du$$

If $\sigma \leqslant \gamma(t, S(t))$, then $(P_\gamma, \alpha_\gamma)$ is a superstrategy and

$$V_{\alpha_\gamma}(T) \geqslant h(S(T)) \text{ and } C_\gamma(0, S(0)) \geqslant E[\frac{h(S(T))}{M(T)}]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

22.1.2 The calibration problem

22.1.2.1 The general idea

22.1.2.1.1 An ill-posed inverse problem In order to fit the volatility surface practitioners use a range of parametric models, such as jump-diffusion models. For every parametric model that one can define, we need to estimate the model parameters from the implied volatility surface. It leads to an ill-posed inverse problem (see Ivanov [1962] and Tikhonov [1963]). It is ill-posed because the inversion is not stable and amplifies market data errors in the solution. To be more precise, letting x be the vector of model parameters and y be the vector of market prices, we want to solve

$$Tx = y$$

where $T : X \to Y$ is a (non-linear) operator between reflexive Banach spaces X, Y, with inverse T^{-1} which is not continuous. We further assume that only noisy data y^δ with

$$||y^\delta - y|| \leqslant \delta$$

is available. The operator T^{-1} can be found by solving a measure among a set of measures, such as

$$||Tx - y^\delta||^2$$

The ill-posedness of the problem means that a small error on market data y can lead to a big error on the solution (model parameters) x. Regularisation techniques allow one to capture the maximum information on x from the set y in a stable fashion (see Section (18.6.1)).

22.1.2.1.2 Description We let $\mathcal{M}(\Psi)$ be an abstract model with model parameters $\psi \in \mathbb{R}^n$, for some $n \in \mathbb{N}$, and consider the pricing map $P : \mathcal{M}(\Psi), \zeta \to \mathbb{R}^m$, for some $m \in \mathbb{N}$, where ζ is the financial product we want to price (market information). We also let $P^*(\zeta) \in \mathbb{R}^m$, $m \in \mathbb{N}$, be the market data corresponding to market information ζ. The calibration problem for a model $\mathcal{M}(\psi)$ is defined by

$$\psi^* = \arg\min_{\Psi} f\big(P(\mathcal{M}(\psi), \zeta), P^*(\zeta)\big)$$

where f is a suitable metric.

22.1.2.1.3 Measures of pricing errors Practically, we choose to estimate implicitly the vector Ψ of model parameters by minimising, given a measure, the distance between the liquid market price $P_t(T_i, K_i)$ and the model price $C_t(T_i, K_i)$ for $i \in I$ where I is the total number of market prices considered. Note, the market price for the ith stock S_i is denoted by $P_i(t)$ while the option price for maturity T_i and strike K_i is denoted by $P_t(T_i, K_i)$. We consider some benchmark instruments with payoff $(H_i)_{i\in I}$ with observed market prices $(P_i^*)_{i\in I}$ in the range $P_i^* \in [P_i^b, P_i^a]$ representing the bid / ask prices. In the option world, we also need to consider a set of arbitrage free model $\mathcal{Q}$ such that the discounted asset price $(\overline{S}_t)_{t\in[0,T]}$ is a martingale under each $\mathbb{Q} \in \mathcal{Q}$ with respect to its own history $\mathcal{F}_t$ and

$$\forall \mathbb{Q} \in \mathcal{Q}\,, \forall i \in I\,, E^{\mathbb{Q}}[|H_i|] < \infty\,, E^{\mathbb{Q}}[H_i] = P_i^*$$

Since the market price P_i^* is only defined up to the bid-ask spread we get

$$\forall \mathbb{Q} \in \mathcal{Q}\,, \forall i \in I\,, E^{\mathbb{Q}}[|H_i|] < \infty\,, E^{\mathbb{Q}}[H_i] \in [P_i^b, P_i^a]$$

Further, different choices of norms for the vector $(P_i^* - E^{\mathbb{Q}}[H_i])_{i\in I}$ lead to different measures for the calibration error. For example, we have

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$
\begin{aligned}
||P^* - E^{\mathbb{Q}}[H]||_\infty &= \sup_{i \in I} |P_i^* - E^{\mathbb{Q}}[H_i]| \\
||P^* - E^{\mathbb{Q}}[H]||_1 &= \sum_{i \in I} |P_i^* - E^{\mathbb{Q}}[H_i]| \\
||P^* - E^{\mathbb{Q}}[H]||_p &= \Big(\sum_{i \in I} |P_i^* - E^{\mathbb{Q}}[H_i]|^p\Big)^{\frac{1}{p}}
\end{aligned}
$$

Obviously we need to choose a measure among this set of measures, and traditionally practitioners consider the Price Norm

$$f_1(i) = \big|P_t(T_i, K_i) - C_t(T_i, K_i; \Psi)\big|^2$$

or the Relative Price Norm

$$f_2(i) = \Big|\frac{P_t(T_i, K_i) - C_t(T_i, K_i; \Psi)}{P_t(T_i, K_i)}\Big|^2$$

22.1.2.2 The problem

22.1.2.2.1 Moments estimation Using market prices, we need to estimate the vector $\psi \in \Psi$ of model parameters in order to price exotic options. This amount to solving the following inverse problem (see Cont et al. [2002]).

***Problem* 1** *Given prices $C_t(T_i, K_i)$ for $i \in I$, find the vector Ψ of model parameters such that the discounted asset price $\widehat{S}_t = e^{-rt} S_t$ is a martingale and the observed option prices are given by their risk-neutral expectations*

$$\forall i \in I\,,\, C_t(T_i, K_i) = e^{-r(T-t)} E^{\psi}[(S(T_i) - K_i)^+ | S_t = S]$$

That is, we need to retrieve the risk-neutral process and not just the conditional densities (European prices) which is equivalent to a moment problem for the process S. If we new option prices for one maturity and all strikes we could compute the risk-neutral distribution of the logarithm price and deduce the model parameters. We would then compute option prices for any other maturity meaning that we can not gain further information from prices of other maturities. However, given market incompleteness, one need to use additional criterion to choose a solution. It means that we need to reformulate the calibration as an approximation problem. For instance, minimising the in-sample quadratic pricing error. Note, it corresponds to the empirical risk minimisation (ERM) problem described in Section (10.1).

22.1.2.2.2 The choice of a volatility model In complete markets, pricing models such as Dupire's formula (see Dupire [1994]), require the knowledge of European call and put prices for all strikes and all maturities. However, in practice we can only observe a few market prices from standard strikes and maturities. For an index, one can easily obtain more than 100 points (prices or implied volatility) where some of them are listed while the others are produced by brokers. On the other hand, there are only a few points for single stock (from 0 to 30) produced only by brokers. These points are not always convex in their strikes due to transaction costs or non-synchronous trading, and as such do not satisfy the no-arbitrage conditions. Further, they can have wide or narrow spreads depending on the liquidity on the market and the volume traded. Thus, observed prices contain measurements errors so that the observed implied volatility is equal to the true volatility plus an error. As a result, the market is incomplete and there are more than one acceptable price (volatility) surface satisfying the no-arbitrage conditions. It is typically an ill posed problem as there may be either no solution at all or an infinite number of solutions. Hence, multiple risk-neutral distributions can fit the option prices, so that one needs some additional criteria to generate a unique probability distribution function (pdf). To do so, one can either impose a functional form to the probability distribution and estimate its parameters using option data, or, one can choose non-parametric methods obtaining perfect fit to market data (see Jackwerth

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1999], Bondarenko [2003] for a review of these methods). However, non-parametric methods are less adapted to the extrapolation problem than the parametric ones, and they tolerate less control over the generated volatility surface. We are therefore going to use a parametric representation of the market call prices in order to smooth the data and get nice probability distribution functions (pdf).

22.1.2.2.3 Accounting for no-arbitrage We saw in Section (22.1.1.5) that Rogers et al. [2010] gave a necessary condition for no-strike arbitrage. These results have provided us with a set of tools and methods to check whether a given volatility parametrisation is free from arbitrage or not. In markets with long maturity products and discrete dividends, it is important for model pricing to obtain a reliable volatility surface satisfying the no-arbitrage constraint not only in space but also in time. One solution is to define a globally arbitrage-free parametric model in time and space.

22.1.2.3 The standard approach

22.1.2.3.1 Using prices Practically, we choose to implicitly estimate the vector $\psi \in \Psi$ of model parameters by minimising, given a measure, the distance between the liquid market price $P_t(T_i, K_i)$ and the model price $C_t(T_i, K_i; \psi)$ for $i \in I$, where I is the total number of market prices considered. The market generally consider the optimisation problem

$$\begin{aligned} \psi^* &= \arg\inf_{\psi} \mathcal{J}(\psi) \\ \mathcal{J}(\psi) &= \sum_{i=1}^{n} w_i \left| P_t(T_i, K_i) - C_t(T_i, K_i; \psi) \right|^2 + \alpha H(\mathbb{Q}, \mathbb{Q}_0) \end{aligned} \tag{22.1.14}$$

where w_i is a weight associated to each market option price $P_t(T_i, K_i)$. The weights are positive and must sum to one. They should reflect our confidence in individual data points which is determined by the liquidity of a given option. The function H is a measure of closeness of the model $\mathbb{Q}$ to a prior $\mathbb{Q}_0$. Many choices are possible for the penalisation function H. To get uniqueness and stability of the solution, the function should be convex with respect to the model parameters. So, the target function is made of two components:

1. penalisation function convex in model parameters ψ
2. quadratic pricing error measuring the precision of calibration

The coefficient α is the regularisation parameter and defines the relative importance of the two terms such that when $\alpha \to 0$ we recover the standard Least Square Error which is no-longer a convex function. If α is large enough, the target function inherits the convexity properties of the penalising function and the problem becomes well-posed. The correct choice of α is important and can not be fixed in advance since its optimal value depends on the data and on the level of error δ one wants to achieve.

22.1.2.3.2 From prices to volatilities The weights in Equation (22.1.14) reflect the relative importance of reproducing different option prices precisely, they must be positive and sum up to one. They may reflect the width of corresponding bid-ask intervals

$$w_i = \frac{1}{|P_t^{bid}(T_i, K_i) - P_t^{ask}(T_i, K_i)|}$$

For options around the money the order of magnitude of the bid-ask is of tens of basis points, that is $< 1\%$. In order to have errors proportional to bid-ask spreads, we must therefore minimise the difference of implied volatilities rather than the difference of prices. As discussed in Section (22.1.1.4), we must invert the Black-Scholes formula

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\Sigma_C(t;T_i,K_i;\psi) = C_{BS}^{-1}\big(C_t(T_i,K_i;\psi);T_i,K_i\big)$$

getting the optimisation function

$$\mathcal{J}(\psi) = \sum_{i=1}^{n} w_i\big|\Sigma_P(t;T_i,K_i) - \Sigma_C(t;T_i,K_i;\psi)\big|^2 + \alpha H(\mathbb{Q},\mathbb{Q}_0) \tag{22.1.15}$$

However, it involves the numerical inversion of the Black-Scholes formula $C_{BS}^{-1}\big(C_t(T_i,K_i;\psi);T_i,K_i\big)$ at each step, which might be very time consuming.
One approach to avoid inverting the BS-formula is to approximate the model implied volatility, getting $\widehat{\Sigma}_C(t;T_i,K_i;\psi)$. An example of such an approach is the SABR model (see Hagan et al. [2002]). In that setting, the optimisation function becomes

$$\mathcal{J}(\psi) = \sum_{i=1}^{n} w_i\big|\Sigma_P(t;T_i,K_i) - \widehat{\Sigma}_C(t;T_i,K_i;\psi)\big|^2 + \alpha H(\mathbb{Q},\mathbb{Q}_0) \tag{22.1.16}$$

Alternatively, given the previous problem we can weight the difference in prices such that its order of magnitude correspond to the difference in volatility. Given a maturity T_i and a strike K_i, we denote the difference at time t between the market price and the model price as

$$\Delta D_t = P_t(T_i,K_i) - C_t(T_i,K_i;\psi)$$

Since the market price can be expressed in terms of the Black-Scholes volatility $\sigma_{BS}(T_i,K_i)$, we can approximate the difference in prices as

$$\Delta D_t \approx \frac{\partial P_t(T_i,K_i)}{\partial \sigma_{BS}(T_i,K_i)}\Delta\sigma(T_i,K_i)$$

where $\Delta\sigma(T_i,K_i) = \sigma_P(T_i,K_i) - \sigma_C(T_i,K_i)$. In that case, the optimisation function becomes

$$\mathcal{J}(\Psi) = \sum_{i=1}^{n} \overline{w}_i\big|\sigma_P(t;T_i,K_i) - \sigma_C(t;T_i,K_i;\psi)\big|^2 + \alpha H(\mathbb{Q},\mathbb{Q}_0)$$

where

$$\overline{w}_i = w_i\Big(\frac{\partial P_t(T_i,K_i)}{\partial \sigma_{BS}(t;T_i,K_i)}\Big)^2$$

which is expressed in volatility. Therefore, we can modify the optimisation in price as

$$\mathcal{J}(\psi) = \sum_{i=1}^{n} w_i\big|\max\big(\frac{\partial \sigma_{BS}(T_i,K_i)}{\partial P_t(T_i,K_i)},100\big)(P_t(T_i,K_i) - C_t(T_i,K_i;\psi))\big|^2 + \alpha H(\mathbb{Q},\mathbb{Q}_0)$$

to re-express it in volatility and where $\max(\cdot,\cdot)$ is used to avoid overweighting of options very far from the money. Put another way

$$\sum_{i=1}^{n}\big|\sigma_P(T_i,K_i) - \sigma_C(t;T_i,K_i;\psi)\big|^2 \approx \sum_{i=1}^{n}\Big|\frac{\partial \sigma_P(t;T_i,K_i)}{\partial P_t(T_i,K_i)}(P_t(T_i,K_i) - C_t(T_i,K_i;\psi))\Big|^2$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

22.1.2.3.3 A global optimum In an incomplete market, a deterministic optimisation method will at best locate one of the local minima of the fitting criterion but will not guarantee the global minima and will not acknowledge the multiplicity of solutions of the initial calibration problem. One solution is to solve the optimisation problem by considering a non-linear programming problem under constraints that do not require computing the gradient of the model. To do so, one can use an improved Differential Evolution (DE) algorithm (see Bloch et al. [2011]) handling constraints in a simple and efficient way.

22.1.3 Trading strategies

A trading strategy is a fixed plan that is designed to achieve a profitable return by going long or short in markets. For every trading strategy, one needs to define assets to trade, entry/exit points and money management rules. Trading strategies are based on fundamental or technical analysis, or both. They are usually verified by backtesting, where the process should follow the scientific method. Technical strategies can be broadly divided into the mean-reversion and momentum groups. Some example of trading strategies are:

- Long/short equity: A long-short strategy consists of selecting a universe of equities and ranking them according to a combined alpha factor. Given the rankings we long the top percentile and short the bottom percentile of securities once every rebalancing period.
- Pairs trade: A pairs trading strategy consists of identifying similar pairs of stocks and taking a linear combination of their price so that the result is a stationary time-series. We can then compute z-scores for the stationary signal and trade on the spread assuming mean reversion: short the top asset and long the bottom asset.
- Trading on the news: The news is an essential skill for astute portfolio management, and long term performance is the technique of making a profit by trading financial instruments (such as stock, currency) just in time and in accordance to the occurrence of events.
- Trading Signals: Trading signal is simply a method to buy signals from signals provider, is a very effective strategy to determine the best time to buy or sell a stock or currency pair. Aggregate analysts forecasts are often used in momentum trading strategies.
- Social trading: Using other peoples trading behaviour and activity to drive a trading strategy.

In general, the development and application of a trading strategy follows eight steps: (1) Formulation, (2) Specification in computer-testable form, (3) Preliminary testing, (4) Optimisation, (5) Evaluation of performance and robustness, (6) Trading of the strategy, (7) Monitoring of trading performance, (8) Refinement and evolution.
A trading strategy can be executed by a trader (Discretionary Trading) or automated (Automated Trading). The latter wraps trading formulas into automated order and execution systems. Advanced computer modelling techniques, combined with electronic access to world market data and information, enable traders using a trading strategy to have a unique market vantage point. A trading strategy can automate all or part of your investment portfolio. Computer trading models can be adjusted for either conservative or aggressive trading styles.
Usually the performance of a trading strategy is measured on the risk-adjusted basis. Probably the best-known risk-adjusted performance measure is the Sharpe ratio. However, in practice one usually compares the expected return against the volatility of returns or the maximum drawdown. Normally, higher expected return implies higher volatility and drawdown. The choice of the risk-reward trade-off strongly depends on trader's risk preferences.

22.2 Supervised learning

Clearly, the calibration problem presented in Section (22.1.2) is a function estimation problem, as described in Section (10.1.1), where the function to learn is Tx. As discussed above, one can either impose a functional form to the probability distribution and estimate its parameters using option data, or, one can choose non-parametric methods

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

obtaining perfect fit to market data. Nonetheless, depending on the chosen model, it can be a long and inefficient process due to the illness of the problem.

We saw in Chapter (10) that with sufficient data, we could train a neural network (NN) to learn the best relationship between some inputs and outputs. As a result, we can use that relationship to interpolate / extrapolate the approximated function. Since neural networks can approximate complex nonlinear relations, they could be used to infer the market prices of options. Several authors, such as Malliaris et al. [1993], Hutchinson et al. [1994], Anders [1996], Yao et al. [2000b], Bennell et al. [2004], proposed to use neural networks as a nonparametric method for estimating the pricing formula of European options as well as their delta (first derivative with respect to the spot price).
Most of these authors focussed on the multilayer perceptron (MLP) (see Section (5.1.3)) and radial basis function (RBF) (see Section (8.2)). However, with new computational advancements, some authors such as Culkin et al. [2017], Stark [2017] decided to assess more complex deep learning network to learn the BS-formula. In most cases the networks are capable of learning the BS-formula up to high level of accuracy. Further, compared with the BS-formula as benchmark, the networks outperform the latter on real market data.
This approach has been extended to more complex option pricing models and proved to be very efficient at reducing the calibration time of these models. As an example of complex models, the rough volatility model (RVM) is a class of continuous-path stochastic volatility models where the instantaneous volatility is driven by a stochastic process with Holder regularity smaller than Brownian motion. They are typically modelled by a fractional Brownian motion (FBM) with Hurst parameter $H < \frac{1}{2}$. As a result, a range of existing pricing models considered as impracticable by practitioners suddenly regained their favour as they can be swiftly calibrated.
Karatas et al. [2019] applied deep neural networks (DNN) for the calibration of European prices in the case of geometric Brownian motion (GBM), variance gamma (VG), both with and without stochastic arrivals. In the latter, we get the GBM with stochastic arrivals (GBMSA) and the VG with stochastic arrivals (VGSA).
We can also use neural networks to learn the map from the model parameters to the shapes of the implied volatility surface. Some recent work on model calibration with neural networks includes Hernandez [2017], Stone [2018], Bayer et al. [2018], McGhee [2018], Horvath et al. [2019], among others.

22.2.1 Fitting vanilla prices

In this section we use machine learning to fit model prices or market prices.

22.2.1.1 Fitting model prices

Several authors, such as Hutchinson et al. [1994], Culkin et al. [2017], considered solving the calibration problem in Equation (22.1.14) with neural networks. However, rather than directly fitting a network to the market prices $P_{t,i}$ they considered the exercise of learning the prices $C_{t,i}$ generated by a particular model. We let $\psi \in \Psi$, $T_i \in [0, T_{max}]$, and $K_i \in [K_{min}, K_{max}]$ for all $i \in I$. Thus, given Equation (10.1.5), the modified optimisation problem becomes

$$\begin{aligned}
\theta^* &= \arg\min_{\theta} \overline{L}(\theta) \\
\overline{L}(\theta) &= \sum_{i=1}^{n} w_i \left|\widehat{F}_{i,t} - C_t(T_i, K_i; \psi)\right|^2 + \alpha H(\theta, \theta_0)
\end{aligned}$$

where $\widehat{F}_{i,t} = h(X_i, \theta)$ is the network with input (or feature) vector X_i and parameter θ, and the labelled output $Y_{i,t} = C_t(T_i, K_i; \psi)$ is the model price with vector of parameters ψ.

***Remark* 22.2.1** *The input vector X_i is the explanatory variables which encompass the market information ζ and the vector of model parameters ψ. That is, $X_i = [\psi, \zeta, T_i, K_i]$ for all $i \in I$.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Thus, once the network has learned the pricing map we get $\widehat{F}(X,\theta^*) \approx P(\mathcal{M}(\psi),\zeta)$. Note, one can select the network of his choice, such as a neural network with a given number of hidden layers, a given number of neurons per layer, and a particular activation function. We can choose any loss function $\overline{L}(\cdot)$ described in Section (10.1.2).

***Remark* 22.2.2** *The network is not build from a no-arbitrage argument (see Section (22.1.1.5)).*

For instance, in the case of the BS-formula, the model parameter is the volatility Σ_{BS}. In the network, the input vector X_i is made of the initial stock price S, the option strike K_i, the option maturity T_i, the interest rate r, and the volatility Σ_{BS}. In the case of a variance gamma model with stochastic arrival (VGSA) the model parameters are the volatility σ, skewness θ, kurtosis ν, rate of mean-reversion κ, the long-term rate of time change η, and the volatility of the time change λ.
Note, model prices can be normalised and expressed with respect to moneyness (see Equation (22.1.9)). In the case of a GBM, one is using the formula in Equation (22.1.10) to price European options. In that setting, the BS-formula satisfies the no-arbitrage condition. As discussed in Section (22.1.1.6), the specific model chosen characterise a particular type of dynamics for the IVS which can be expressed in terms of the strike price and the underlying spot price. That is, the smile can either be expressed in terms of moneyness, or in terms of absolute value for the strike and spot prices. The former is generated by space homogeneous models and the latter by space inhomogeneous models. Thus, the price normalisation should be treated with caution.

22.2.1.1.1 The tests Hutchinson et al. [1994] considered radial basis function network, multilayer perceptron (MLP) network and projection pursuit, and generated (with MC) a two-year sample of daily stock and option prices, called the training path. Several paths were generated to obtain a measure of success of the average network pricing formula. The generation of option prices is based on the CBOE rules where the strike prices are multiples of $\$5$ for stock prices in the range $[25, 200]$. When options expire a new maturity date is introduced, and the two strike prices closest to the current spot price are used. On average there are 81 options per sample path with a total of 6001 data points. It corresponds to six months of futures options on the $S\&P$ 500 (see below). To reduce the number of variables, options are expressed in terms of the moneyness. Two inputs are used: moneyness and time to maturity. The quality of prices and delta-hedging (one day time-step) is assessed and performances measures are the R^2 and the tracking error for the delta hedging performance. An independent testing period of six months is chosen.

Culkin et al. [2017] trained a multilayer feed-forward network to learn the BS-formula. The chosen variables are moneyness, the option maturity, the spot rate, continuous dividend, and the volatility. In order to generate sufficient data for the network to learn the input-output relationship, they defined a range for each of the component of the input vector and computed the model prices by using the BS-formula in Equation (22.1.7). Using a four hidden layer network, they simulated $300,000$ call option prices and obtained $R^2 = 0.9982$ in the linear regression against the BS-formula in the out of sample.

22.2.1.2 Fitting market prices

As discussed in Section (22.1.2), we want to use a network to solve the calibration problem in Equation (22.1.14) on real market prices $P_{t,i}$. The optimisation problem becomes

$$\begin{aligned}\theta^* &= \arg\min_{\theta} \overline{L}(\theta) \\ \overline{L}(\theta) &= \sum_{i=1}^{n} w_i \left|\widehat{F}_{i,t} - P_t(T_i,K_i)\right|^2 + \alpha H(\theta,\theta_0)\end{aligned}$$

where $\widehat{F}_{i,t} = h(X_i,\theta)$ is the network with input (or feature) vector X_i and parameter θ, and the labelled output $Y_{i,t} = P_t(T_i,K_i)$ is the market price. In that setting, the input vector X is made of the stock price S, the option strike K, the option maturity T, and the interest rate r.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

***Condition* 1** *Given Remark (22.2.2), one must make sure that the market prices are arbitrage-free in time and space.*

22.2.1.2.1 The tests Hutchinson et al. [1994] also tested their approach on the $S\&P$ 500 futures option market and used the BS-formula as a benchmark with a volatility statistically estimated on a sliding window of 60 days. The volatility is estimated as follows:

$$\hat{\sigma} = \frac{s}{\sqrt{60}} \tag{22.2.17}$$

where s is the standard deviation of the 60 most recent daily returns. The data cover the five year period from Jan87 to Dec91, the options have quarterly expirations with a total of 40 to 50 points on a typical day. The data is divided into 10 nonoverlapping six-month subperiods, which gives an average of 137 call options and a total number of points of 6246 for a subperiod. A separate learning network is trained on each of the first 9 subperiods and tested on the following subperiod. That is, we keep changing model for each subperiod, or equivalently we re-calibrate the model every six month.

Stark [2017] applied the same methodology on options written on the DAX 30 from 2013 to 2017 and obtained consistent results with previous literature. The dataset contains 668 different call options with 25 different strikes, giving a total of 130132 points. The data has been filtered, options with maturity larger than two years were discarded, moneyness is in the range $[0.7, 1.20]$, which resulted in 48486 points remaining. The total number of days is 1231, the training set is 785 days and the testing set is 424 days. The model is an MLP with four hidden layers and softplus activation function. There are two inputs: the moneyness and time to maturity. Rates and volatility are estimated as in Hutchinson et al.

22.2.1.2.2 Results In general the NNs do better for out-of-the money (OTM) because BS-formula overprices these options. While both NNs and BS-formula underprice in-the-money (ITM) options, the former outperform the latter. Note, near-the-money (NTM) options is not as good as the wings for NNs. When partitioning with respect to maturity NNs are superior to the BS-formula for medium and long term maturities, except for short term maturities at all moneyness level. In that case the NNs overprice the short-term NTM and OTM options. The longer the maturity, the better for the NNs.
When it comes to assessing the delta it seems the NNs outperform the BS-formula for OTM options, but the reverse is true in the case of ITM options. This is not surprising given Remark (22.2.2).

22.2.1.2.3 Analysis and solutions Some of the reasons explaining these results are:

- The BS-volatility in Equation (22.2.17) is an estimate of the implied ATM-volatility for short maturities. So naturally the NTM options with short maturities are well approximated with the BS-formula. As the time to maturity increases and one consider the wings of the surface this volatility no-longer matches the IVS, explaining the poor results of the BS-formula.
- It has been assumed that the distribution of the returns is independent of the level of the spot price, so that the pricing formula is homogeneous of degree one both in the spot level and strike price. However, as discussed in Section (22.1.1.6), it could be homogeneous, inhomogeneous, or anything in between.
- The pricing function is not smooth near-the-money.
- The approximation of the derivative of the pricing formula require to impose smoothness constraints.
- Even though neural networks are function approximator (see Remark (10.1.1)), it does not guarantee that the derivatives of the function are also learned. Need to impose smoothness constraints on the loss function.

Some solutions to solve these problems are:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- A measure of performance for validating the capacity of a network to reproduce market prices should be to check that the option prices generated in the testing sample are within the market bid-ask.
- One should relax the homogeneous assumption and consider the spot level and strike price as variables.
- To refine the granularity of the variables, especially around the money, where the delta goes from convex to concave shape. Thus, if we want to cover large range of strikes we need to massively increase the size of the data.
- Alternatively, one can keep coarse points but learn the delta of the market prices.

This is what we are going to discuss in the next section.

22.2.1.3 Completing market prices

22.2.1.3.1 The problem Contrary to fitting a theoretical model, at a given time t, we only observe a finite number of market prices (usually very small). Even when considering option prices at different time t, Hutchinson et al. [1994] only gathered around 6000 options on the $S\&P$ 500 over a period of six months. This is more than unsufficient to train a complex multilayer model. Further, chances are that the spot price S does not vary too much over such a period, so that we can only observe prices around the moneyness (see Equation (22.1.9)). However, over a longer period of time the moneyness will vary and possibly jumps (October 1987 crash, subprime crisis in 2008). Acknowledging this problem, Hutchinson et al. [1994] considered changing the learning network every six month. However, in order to get good learning results without changing (or recalibrating) the network, the sample under consideration should span all the domain of definition. That is, it should cover the moneyness in a range of say $[0.1, 2.0]$ for all liquid maturities. Put another way, since we do not know the joint probability distribution $P(X, Y)$, we need to generate a large number data spanning all the domain of definition, that is, European call and put prices for all strikes, all maturities, and all spot level.

22.2.1.3.2 Generating arbitrage-free data Unfortunately there is no exact model we can use to generate sufficient market prices. Nonetheless, to generate a continuum of market prices, we can use a parametric representation of the market call prices that smooth the data and get nice probability distribution functions (pdf). However, we saw in Section (22.1.1) that we needed to introduce necessary conditions to obtain an implied volatility surface (IVS) free from static arbitrage. One solution is to define a globally arbitrage-free parametric model in time and space capable of reproducing a large range of implied volatility surfaces. To do so, Bloch [2010] proposed to impose smoothness and value constraints directly on the market prices and their resulting implied volatility surface. He considered a parametric representation of the market call prices under constraints called MixVol model (see Appendix (22.7)) in order to smooth the data and get nice probability distribution functions (pdf). A differential evolution algorithm (see Bloch et al. [2011]) was used to calibrate the model's parameters to a finite set of option prices. We can therefore fit the existing market prices with this model and use it to interpolate / extrapolate market prices in time and space in an arbitrage free way.
However, if we want to produce hundred of thousands of market prices for all strikes, all maturities, and all spot level we would need a very large history of option prices to calibrate. Alternatively, if we had an intuitive parametric representation of the smile allowing us to manually modify its shape when the underlying price moves, while preserving the no arbitrage in time and space, we could generate sufficient data. This is precisely what we do in the MixVol model when we run the scenarios analysis described in Appendix (22.7.3). In addition, as described in Appendix (22.7.2), we can compute analytically the Greeks of the model. Thus, we can generate:

- European call and put prices for all strikes, all maturities, and all spot level.
- Their associated Greeks

As a result, we are guaranteed to generate sufficient data to properly train a deep network to learn both market prices and their associated Greeks, while satisfying the Condition (1).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

22.2.2 Calibration

As discussed in Section (22.1.2), model calibration methods rely on the repetitive evaluation of the map from model parameters ψ to the BS implied volatility Σ_{BS}, which is very time consuming in the case of complex pricing models. For instance, in the BS-model, there is only one parameter to fit, the volatility Σ_{BS}, but in the Heston model (see Heston [1993]), the model parameters are the initial variance v_0, the speed of mean reversion $\hat{v}$, the volatility of volatility ξ, and the correlation ρ. That is, $\psi = \{v_0, \hat{v}, \xi, \rho\}$.
Since neural networks (NNs) can be trained off-line, they should allow for the calibration of more complex models by speeding up the computational time. That is, the approach described in Section (22.2.1.1) can be extended to complex option pricing models such as a stochastic volatility model or a jump-diffusion model. The idea being to shift the lengthy numerical training process into an off-line preprocessing task, reducing the online time for option pricing. Since neural networks are function approximator (see Remark (10.1.1)) they can use off-line methods to learn complex pricing models.
Contrary to Section (22.2.1), where we used machine learning to fit model prices, we can directly learn the model parameters from the model prices. We can then use the trained network in the classical calibration process with a simple gradient descent or a genetic algorithm to estimate the model's parameters $\psi \in \Psi$. Thus, rather than using a Monte Carlo engine or a partial differential equation for pricing options, which is time consuming, we use the trained network to interpolate the prices.

22.2.2.1 Description

22.2.2.1.1 An overview Hernandez [2017] trained a network to directly return calibrated parameters of a stochastic volatility model by learning the inverse map (calibration routine in Equation (22.1.15)). Here, option price approximation and parameter calibration are performed within the same network. The network was calibrated on rate contracts using the short rate model of Hull et al. [1990]. In the case of equity market, the input data is the implied volatility surface. Hence, making an analogy with an image, it has been suggested to use a convolutional neural network (CNN) (see Section (9.1.1)). Following that route, Stone [2018] calibrated a rough volatility model (RVM) with a convolutional neural network. Following a different approach, Bayer et al. [2018] used a fully connected neural network regression to approximate the volatility map φ for the Heston model and rough Bergomi volatility model. In general, the volatility map and its associated Jacobian do not have an analytical form so that we can rely on a numerical approximation for the model price, leading to an approximate implied volatility map $\hat{\varphi}$. The NN is trained to learned the latter, leading to the representation $\varphi_{NN} : \mathcal{I} \to \mathbb{R}_+$
Liu et al. [2019] proposed to use one neural network for pricing options with a Heston model, and use this result with another network for learning the relationship between implied volatilities and option prices.
Another approach is to separate the calibration procedure in two parts:

1. First learn the pricing map via a neural network, and store this map during the off-line training. That is, get $\hat{F}(X, \theta^*) \approx P(\mathcal{M}(\psi), \zeta)$.

2. Then, solve the calibration problem (on-line) for $\psi \in \Psi$ with the learned price map.

Several methods have been proposed to tackle this approach: McGhee [2018] considered a pointwise training, Horvath et al. [2019] focussed on an image-based and implicit training.

22.2.2.1.2 Relationship between implied volatilities and option prices Liu et al. [2019] proposed to use one neural network for pricing options with a Heston model, and use this result with another network for learning the relationship between implied volatilities and option prices. A generator creates the data to train the network and a predictor is produced to approximate the option prices with the trained network. A COS method (using Fourier-cosine series expansions) is used to generate the model prices and an iterative root-finding Brent method is used to generate the implied volatilities. Among the market information, the moneyness is chosen. Latin hypercube sampling (LHS) is used to generate random samples of the parameter values. In the Black-Scholes model, in order to learn the implied

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

volatility, the model prices and market information $\{V, S_0, K, \tau, r\}$ are passed as inputs and associated implied volatilities σ are passed as outputs. In the Heston model, the input parameters are $\{r, \rho, \overline{\nu}, \gamma, \nu_0\}$ and the market information is (τ, m), where $m = \frac{S_0}{K}$ is the moneyness. Once the prices have been generated with the COS method, the Brent's method is used to compute the implied volatility Σ_{BS}. The AI approach follows two steps: the Heston-ANN is used to compute the option prices, then an IV-ANN is used to compute the corresponding implied volatilities. The larger the range of parameters the worth the performance of the network.
Since the BS-model and the Heston model are homogeneous models (see Section (22.1.1.6)), to capture the relationship between implied volatilities and option prices, the data only need to be generated at inception (t_0, S_0). Since moneyness is considered and prices are normalised, it is assumed that as time passes that relation is stationary. This is no-longer true for inhomogeneous models and for real market data. In that case we need to consider a time-series of prices and associated implied volatilities.

22.2.2.1.3 Pointwise learning Going further, some authors considered approximation method for option prices as well as implied volatility. The aim being to directly estimate the model parameters $\psi \in \Psi$ by mapping these parameters to an approximation of the option prices or the implied volatility surface, respectively. Pointwise learning consists in learning one pair (T_i, K_i), for $i \in I$, for a fixed vector of model parameter ψ_i at a time. This way, the approximated points implied volatility $\widehat{\Sigma}_t(T_i, K_i; \psi)$, for all i, can be learned with a neural network. Some authors such as Bayer et al. [2018], McGhee [2018], among others, considered the pointwise learning method.
We let $\psi_i \in \Psi$, $T_i \in [0, T_{max}]$, and $K_i \in [K_{min}, K_{max}]$ for all $i \in I$. In that setting, learning the volatility map, the modified optimisation problem becomes

$$\begin{aligned} \theta^* &= \arg\min_{\theta} \overline{L}(\theta) \\ \overline{L}(\theta) &= \sum_{i=1}^{n} w_i \left| \widehat{F}_{i,t} - \widehat{\Sigma}_t(T_i, K_i; \psi_i) \right|^2 + \alpha H(\theta, \theta_0) \end{aligned}$$

where $\widehat{F}_{i,t} = h(X_i, \theta)$ is the network with input (or feature) vector X_i and parameter θ, and the labelled output $Y_{i,t} = \widehat{\Sigma}_t(T_i, K_i; \psi_i)$ is the model approximated IV for the pair (T_i, K_i) with parameter ψ_i. Thus, once the network has learned the volatility map, we get $\widehat{F}(X, \theta^*) \approx P(\widehat{\Sigma}(\psi), \zeta)$, where $\Sigma(\psi) = \{\widehat{\Sigma}_t(T_i, K_i; \psi_i);\ i = 1, ..., n\}$.
We define the market implied volatility as $\Sigma_{BS} = \{\Sigma_{BS}(T_i, K_j);\ i = 1, ..., N_T; j = 1, ..., N_K\}$. Setting $\widehat{F}_t(\psi; T, K) = \widehat{F}_t(X, \theta^*; T, K)$, we can the solve the calibration problem as

$$\psi^* = \arg\min_{\psi \in \Psi} \sum_{i=1}^{N_T} \sum_{j=1}^{N_K} \left| \widehat{F}_t(\psi; T_i, K_j) - \Sigma_{BS}(T_i, K_j) \right|^2 + \alpha H(\psi, \psi_0)$$

This method requires calibrating the IVS in time and space by considering the pair (T_i, K_j) for all $i = 1, ..., N_T$ and $j = 1, ..., N_K$. Note, in some cases the smiles are calibrate maturity by maturity, or over a range maturity. For example, McGhee [2018] calibrated the SABR stochastic volatility model with a neural network. The chosen network is a one hidden layer with a sigmoid activation function in order to apply the universal approximation theorem (see Section (5.1.3.3)). The off-line learning is done with the approximation scheme (see Hagan et al. [2002]) in the case $\beta = 1$. Two networks are considered one for the maturities $T \in [1D, 1Y]$ and one for the maturities $T \in [1Y, 2Y]$. The first and second derivatives computed with finite difference schemes are also analysed. The network is also assessed against more accurate numerical schemes, the SABR integration scheme and the two factor finite difference scheme SABR 2F-PDE.

22.2.2.1.4 Image-based implicit learning By analogy to digital image (see Section (1.4.2)), some authors proposed to express the implied volatility surface (IVS) as an image represented by pixels. This is called the image-based representation of the IVS. Contrary to the pointwise learning, where we learn one pair (T_i, K_i), for $i \in I$, for a fixed

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

vector of model parameter ψ_i, at a time, in the IBI learning, we want to learn the model volatility map $\psi \to \Sigma(\psi)$ for $\psi \in \Psi$. The idea being to learn the full IVS (and not a pair) at a time.
For instance, Horvath et al. [2019] let $\Delta = \{(T_i, K_j)i = 1, ..., N_T; j = 1, ..., N_K\}$ be a fixed grid of maturities and strikes. In that setting, learning the volatility map, the modified optimisation problem becomes

$$\begin{aligned}\theta^* &= \arg\min_{\theta} \overline{L}(\theta) \\ \overline{L}(\theta) &= \sum_{i=1}^{n}\sum_{p=1}^{N_T}\sum_{q=1}^{N_K} w_i\big(\widehat{F}_{i,t}(p,q) - \widehat{\Sigma}_t(T_p, K_q; \psi_i)\big)^2 + \alpha H(\theta, \theta_0)\end{aligned}$$

where $\widehat{F}_{i,t}(p,q) = h(X_{i,p,q}, \theta)$ is the network with input (or feature) vector $X_{i,p,q}$ and parameter θ, and the labelled output $Y_{i,t} = \widehat{\Sigma}_t(\psi_i)$ where $\widehat{\Sigma}_t(\psi_i) = \{\Sigma_t(T_p, K_q; \psi_i);\ p = 1, ..., N_T; q = 1, ..., N_K\}$ is the model approximated IV with parameter ψ_i.
Setting $\widehat{F}_t(\psi) = \widehat{F}_t(X, \theta^*)$, we can the solve the calibration problem as

$$\psi^* = \arg\min_{\psi\in\Psi} \sum_{i=1}^{N_T}\sum_{j=1}^{N_K} \big|\widehat{F}_t(\psi; T_i, K_j) - \Sigma_{BS}(T_i, K_j)\big|^2 + \alpha H(\psi, \psi_0)$$

The author argue that the optimum parameter depends implicitly on Δ so that we should write $\widehat{F}_t(\psi) = \widehat{F}_t(X, \theta^*(\Delta))$ justifying the implicit learning.

22.2.2.2 Problems

As discussed above, the main idea for using neural networks in the calibration of complex option pricing models is to shift the lengthy numerical calibration process into an off-line preprocessing task. Since neural networks are function approximator (see Remark (10.1.1)) they can use off-line methods to learn complex pricing models. Once trained, the network is just a deterministic interpolator / extrapolator. We can then use the trained network in the classical calibration process.
We have denoted ψ the vector of model parameter and ζ the information on the financial products. The calibration process (see Section (22.1.2)) has been expressed into finding the vector ψ such that the model implied volatility surface $\Sigma_t(\psi)$ is as close as the market one. Further, the volatility map (see Definition (22.1.4)) is expressed in terms of the Black-Scholes volatility. In that setting, at time t, market information for a call option consists of the spot price S_t, the spot rate r, the repo rate r_f, the strike K, and the maturity T. That is, $\zeta = \{S_t, r, r_f, K, T\}$.
In most articles mentioned above, the implied volatility is expressed with respect to the pair (T, K) or a fixed grid of size $N_T \times N_K$. While the neural networks are used to learn the volatility map offline, this is done for $\{S_t, r, r_f\}$ fixed. As discussed in Section (22.1.1.6), the dynamics of implied volatility surface (IVS) are neither stationary nor Markovian. In the case of homogeneous models, we can express the volatility surface in term of moneyness. This is no-longer the case for inhomogeneous models such as universal models and the SABR model with $\beta \neq 1$. In that case, having the same vector of parameter ψ will produce different volatility surface for different spot level.
The calibration process is strongly dependent on the change in spot price ΔS over a period of time. In a regime of high volatility with large spot movements, traders will want to frequently recalibrate. However,the underlying can take an infinity of values. One possibility would be to store a range of calibrated network for some discrete values of the underlying at fixed interval ΔS and then interpolate. An alternative solution would be to introduce the spot price S in the input vector X of the network. So, rather than learning an image we would need to learn a cube for (S, T, K). The dimension would further increase should we introduce the spot rate and repo factor in the input vector.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

22.2.3 Fitting exotic prices

22.2.3.1 Description

Some authors suggested using machine learning to fit exotic prices from models for calibration purposes as well as replacing Monte Carlo simulations when inputs change. In practice, complex models are calibrated on vanilla options. Nonetheless, in some cases, some exotic prices could be added to the set I of market prices considered. More importantly, since they can be trained off-line, the neural networks (NNs) should allow for the use of more complex models when pricing exotic options. NNs could also be used for risk calculation.
For instance, Karatas et al. [2019] trained a deep network to fit barrier and American prices. Given some range for the explanatory variables, a training set I of market prices is generated, and a set of deep neural network is tested. Selection is based on obtaining the smallest mean-squared-error (MSE) (or largest R^2) within the training set. One out-of-sample test is performed keeping everything fixed and varying moneyness. Brostrom et al. [2018] investigated the use of neural networks for pricing rainbow options against a second-order Taylor polynomial. The sensitivities are: Delta, Gamma, Vega, Volga, Rho and second-order Rho. Note, each sensitivity was numerically computed for each underlying asset. Here the data set consisted in 1 million samples with different combinations of all inputs in random order. The data set was split in a training set (80%) and a validation set (20%). New data was generated to evaluate the performance of the networks. Different ANNs were tested on the same validation data set and selection was obtained based on MSE, absolute percentage error (MAPE) and mean percentage error (MPE). Babbar et al. [2019] proposed a deep learning approach to exotic option pricing under local-stochastic volatility (LSVol). The model is tested on barrier options. As discussed in Section (22.1.1.6), the LSVol is a universal model solved with a two-factor ADI finite difference scheme or a Monte Carlo engine. It is a space inhomogeneous model where the spot, strike, and barrier level should be considered independently. This is not the case, as a ratio barrier over spot is considered. Further, the set of market information (excluding the barrier level) is fixed and the set of model parameters (including the barrier level) is varied

22.2.3.2 Problems

In order to speed up computation time of exotic prices, some authors considered learning these prices with a neural network. However, there are no closed-form solutions for these prices and one must rely on numerical approximations. However, several problems arise when fitting approximated exotic prices and we will present some of them.

22.2.3.2.1 Data A Monte Carlo engine generate random numbers X_i from a probability density function $f_X(x)$ and compute an objective function for each simulation. Given $Y = h(X)$, then we get

$$E[Y] = E[h(X)] = \int_{-\infty}^{\infty} h(x) f_X(x) dx$$

with sample average

$$\widehat{\mu} = \frac{1}{n} \sum_{i=1}^{n} h(x_i)$$

where n is the sample size and x_i is an independent sample of the random variable X. By the strong law of large numbers, $\widehat{\mu}$ converges almost surely to $E[Y]$

$$\widehat{\mu} = \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} h(x_i) \to \int_{-\infty}^{\infty} h(x) f_X(x) dx = E[Y]$$

The variance is given by

$$Var(\frac{1}{n} \sum_{i=1}^{n} h(x_i)) = \frac{\sigma^2}{n} \text{ where } \sigma^2 = Var(x_i)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

So the sample variance is

$$\widehat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(h(x_i) - \widehat{\mu}\right)^2$$

and

$$\sigma_{\widehat{\mu}} \propto \frac{\widehat{\sigma}}{\sqrt{n}}$$

which states that the standard error $\sigma_{\widehat{\mu}}$ is proportional to $\frac{1}{\sqrt{n}}$.
Hence, the prices generated by a Monte Carlo engine are approximation to the true model prices where the error is directly linked to the numerical settings of the engine. In our notation, $Y_{i,t} = C_t(T_i, K_i; \psi)$, $i = 1, ..., n$, are the true model prices and $\widehat{Y}_{i,t} = \widehat{C}_t(T_i, K_i; \psi)$, $i = 1, ..., n$, are the approximate Monte Carlo model prices defined as

$$\widehat{Y}_{i,t} = Y_{i,t} + \epsilon \text{ , } i = 1, ..., n$$

where ϵ is some noise.
In that setting, the optimisation problem becomes

$$\begin{aligned} \theta^* &= \arg\min_{\theta} \overline{L}(\theta) \\ \overline{L}(\theta) &= \sum_{i=1}^{n} w_i \left|\widehat{F}_{i,t} - \widehat{C}_t(T_i, K_i; \psi)\right|^2 + \alpha H(\theta, \theta_0) \end{aligned}$$

where $\widehat{F}_{i,t} = h(X_i, \theta)$ is the network with input (or feature) vector X_i and parameter θ, and the labelled output $\widehat{Y}_{i,t} = \widehat{C}_t(T_i, K_i; \psi)$ is the approximated model price with vector of parameters ψ.
As discussed by Bloch [2019], while neural networks are very good at calibrating and forecasting the Mackey-Glass time series, the performance is downgraded when adding a noisy component to it. Thus, it seems very difficult for any neural network to learn the true model prices $C_t(T_i, K_i; \psi)$, $i = 1, ..., n$. One solution would be to filter the Monte Carlo prices with wavelet tools or other filtering techniques in view of inferring the true model prices.

22.2.3.2.2 Labelling the data A training set I of complex model prices is generated, usually with a Monte Carlo engine. Nothing is said about the numerical settings, convergence of the pricer, and the type of errors produced by the approximation. It would make sense to test that the pricer recover the forward price, the put-call parity is satisfied on European options for large enough maturities and that the corresponding density integrates to one. In the case of barrier options, it would make sense to test convergence against a binomial tree with sufficient time steps. We do not want the network to learn noisy prices that would not satisfies elementary no-arbitrage conditions. This is a difficult problem, because it implies using different numerical settings for the Monte Carlo engine, in particular, when the model considered is a multifactor model. In the case of knockin and knockout events, both discrete and continuous, it would be relevant to explain how the barrier treatment has been treated (see Glasserman [2004]).

22.2.3.2.3 Defining the range for the explanatory variables When training a neural network to the set of prices I, one has to define a range for the market information $\zeta = \{S_t, r, r_f, K, T\}$ as well as for the model parameters ψ. Several problems occur. The range for the moneyness is generally chosen around-the-money no matter what is the maturity. In a diffusion model, we should account for the cone of diffusion and probably scale that range with T^α for α near $\frac{1}{2}$. Similarly, the range for the model parameters is too narrow, considering that the IVS changes over time, and that in some market conditions, no model can fit exactly the IVS. This result in model parameters being stretched to extreme values. Further, one can not randomly select the vector ψ from the set Ψ as some model combinations do not make sense. For instance, in the Heston model (see Heston [1993]), for the variance to remain positive, the parameters must satisfy the Feller condition (see Feller [1951]).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

22.2.3.2.4 Measuring performance As previously discussed, given some narrow range for the explanatory variables, a training set I of market prices is generated. A set of networks with varying number of layers and varying number of neurons per layer is tested on the training set. However, it is not very relevant to select the best network from the training set. One need to test the network for its capacity to interpolate and extrapolate the data. As explained earlier, the dynamics of the IVS are non-stationary so that one has to consider all the combinations of market information and model parameters. It would make more sense to generate a very large set I of market prices obtained from spanning the sets of market information ζ and model parameters Ψ with the largest possible ranges. We could then define a collection of training sets I_{Train}, from coarse to fine granularity, and a non-overlapping collection of testing sets I_{Test}. Given a training set, we would then measure performance in the latter. The idea being to identify both the granularity of the training set and the network producing the best results on the testing sets.

22.2.3.2.5 Additional parameters As previously explained, there are no closed-form solutions to exotic prices and one must rely on numerical approximations. As a result, neural networks can not directly learn the true model prices but only some noisy approximations which depend on some additional parameters related to the numerical settings of the Monte Carlo engine. Further, to improve convergence of the Monte Carlo engine we might apply numerical techniques involving extra parameters. such as control variates. In addition, when pricing a particular exotic option with a Monte Carlo engine, we rely on some additional parameters. In the case of barrier option, one need to modify the algorithm to account for the fact that the continuous process has been discretised. That is, between two discrete dates we need to estimate the probability of crossing the barrier. We let the vector ξ represents all the numerical parameters linked to the Monte Carlo engine as well as the ones associated to a particular exotic option. Thus, the approximated prices are denoted $\widehat{C}_t(T_i, K_i; \psi, \xi)$, $i = 1, ..., n$. At the moment most articles ignore the vector ξ, or, at best keep it fixed when labelling the data in the training and testing samples.

22.2.3.2.6 One network per payoff When pricing exotic options in investment banks, a model is chosen, and its parameters ψ are calibrated on liquid vanilla options. If we price a barrier option with a Monte Carlo engine, some trajectories of the processes are evolved, the events are accounted for, and the sample average is computed. If we modify a payoff or some events, the model is not recalibrated and the same trajectories are evolved. Only the events will be modified in the Monte Carlo engine. However, when learning exotic prices with a neural network, we will change the network if we modify the payoff. For example, in the case of an up-and-out (UO) barrier option and down and up-and-in (UI) one with the same strike and expiration, the payoff will be modified in the Monte Carlo framework, but the core engine will remain the same. However, when using neural networks to learn these prices, we may end up with two very different networks having different number of layers and not the same number of neurons per layer. Several problems arise. The trading desk is going to have to manage a very large number of networks. In addition, there is an in-out parity for barrier options given by

$$C(t; T, K) = C_{in}(t; T, K) + C_{out}(t; T, K)$$

where $C(t; T, K)$ is the price of a vanilla option. When using neural networks, we may end up with three very different networks not guaranteeing the above in-out parity. However, when using the same core Monte Carlo engine this parity is preserved.

22.2.3.3 Risk analysis

22.2.3.3.1 Events In mostly all articles on using neural networks for pricing vanilla or exotic options, the authors put forward their property of universal interpolators. As stated from the universal approximation theorem (see Section (5.1.3.3)), they rely on their ability to represent any smooth multidimentional function. The function should be of class C^∞. However, contingent claims are far from smooth. Most events in option pricing are not continuously differentiable. For instance, a call option is continuous but not differentiable at the strike K when at maturity T. Hence, it is of class C^0 but not of class C^1. Obviously, it is smoothed with time, as the time to maturity increases. However, in the case of exotic options some events can occur every day, every week, or every month. The role of risk

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

is to analyse the change in PnL near such an event. For instance, in the case of barrier option, we want to know how the network would map the prices when the underlying asst is near the barrier. It makes it very difficult to get a generic neural network for barrier options. In a universal model where the spot, the barrier and the strike have to be considered independently, the number of combinations corresponding to possible scenarios is huge in a discrete domain.

22.2.3.3.2 Extreme market behaviours When performing risk analysis, we are interested in extreme market behaviours such as large spot movements and very skewed volatility surfaces. In that case, the choice of the domain of definition, both for market factors and model parameters should be as large as possible. In the case of a complex model with many parameters and when considering exotic options such as rainbow options, the domain is a hypercube and the labelling of the set I is very expensive.

22.3 Reinforcement learning

We discussed in Section (22.2) the use of supervised learning to learn option prices in view of getting a fast approximator to interpolate or extrapolate model prices. We are now going to show how to use reinforcement learning (RL) to price option in a discrete-time framework, accounting for the intrinsic risk.
In classical mathematical finance, the fair price of an option is related to the optimal value function of a Markov decision process (MDP). Solutions are rooted in dynamic programming and require explicit knowledge of the transition probabilities (see Ben-Ameur et al. [2007]). In general, we assume the transition matrix and the expected next step reward to be known and use dynamic programming (DP) to solve the problem.
As discussed in Section (15.1), in reinforcement learning (RL), state space and transition probabilities are implicit as the learning is based on trajectory samples from the MDP. As a result, we do not need to specify a price process and can directly learn the pricing model from the data. This is done by considering a sequential risk minimisation in a Markov decision process (MDP) problem over fixed-time horizon. In that case the agent learns from data generated from an option trading strategy.

Grassl [2010] formulated derivatives pricing in terms of reinforcement learning problem. He devised a simple Markov decision process where the optimal value function is equivalent to the fair price of an option.
Gosavi [2010] proposed the construction of a risk-sensitive MDP by adding a one-step variance penalties to a finite-horizon risk-neutral MDP problem (see Definition (15.3.4). He used a variance-adjusted value as a compensation for the risk in the objective (see details in Section (17.2.2)). In the same spirit, assuming normal returns, Halperin [2017] proposed to follow a multi-period version of the mean-variance approach of portfolio selection (see Appendix (28.1.1)) to derive a discrete BS-formula in incomplete market. He incorporated a quadratic reward into a standard MDP formulation, obtaining the QLBS model. Interestingly, the continuous-time BS-model is argued to be the continuous-time limit of a model-based RL (in the case of normal returns), where all the data needed are: the current stock price and the volatility. In the discrete case, the pricing formula requires a large quantity of data.
The Q-function learning algorithm was used to compute the option price in the cases where the transition probabilities are unknown. Halperin [2018] provided a numerical Q-learning analysis of his model.

The method proposed by Halperin [2017] is a model-based approach, where the stock price is taken under the historical measure P. A local risk minimisation is performed by considering a backward recursive equation for the replicating portfolio. However, it is not consistent with the self-financing condition since the variations of the replicating portfolio are not only due to the variations of the assets (see Appendix (22.5)). Market is implicitly assumed incomplete since the variance of the portfolio is minimised. One can show that it is similar to accounting for the quadratic risk of the cost process introduced by Follmer et al. [1986] and Schweizer [1988]. In that setting, we show that one can not work under the historical measure P and assume non-null interest rates when estimating risk in the theory of portfolio replication. It leads to erroneous results and misleading interpretation. The proper solution consists in deriving a forward recursive equation for the replicating portfolio, and, taking the money market as numeraire, work under an equivalent martingale measure, where discounted prices are martingale.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Further, we extend this approach by considering risk for non-normal market returns and show that the reward depends on the variance of the portfolio as well as its skewness and kurtosis. We are going to describe these approaches in light of the results on RL (see Chapter (17)) as well as the CAPM (see Appendix (28.1.1)).

22.3.1 Pricing options

Grassl [2010] formulated derivatives pricing in terms of reinforcement learning problem. He devised a simple Markov decision process (MDP) where the optimal value function is equivalent to the fair price of an option. We are going to briefly describe this approach.

22.3.1.1 Formulation

Using notation in Section (22.1.1), we consider the option on the underlying X with maturity T and price given in Equation (22.1.3). We assume the trader holds one unit of an option C at time t. For all $t < T$, the trader can either

- sell C at the price C_t, or
- hold C. If he holds the option at maturity, the option is executed (sold) for $C_T = h(X_T)$

The trading episode ends when the option is sold or executed. Contrary to portfolio optimisation (see Appendix (28.3)), where the strategy is the portion of wealth invested in each asset, in option pricing the strategy to either sell or hold the option. In that setting, the action space $\mathcal{A}$ consists of only two choices:

$$a_t = \begin{cases} a_S \text{ if the option is sold} \\ a_H \text{ otherwise} \end{cases}$$

***Assumption* 22.3.1** *It is assumed that the market state m_{t+1} is independent from a_t. Only the trader position δ_{t+1} depends deterministically on his action.*

Combining m_t and δ_t yields the state of the world at time t, that is, $s_t = (m_t, \delta_t)$. Here, a transition from m_t to m_{t+1} provides information for the choices a_S and a_H. Thus, only the transitions of the market states m_t need to be observed.
The state-action reward is taken to be the cash flows

$$R(s_t, a_t) = \begin{cases} C_t \text{ if } a_t = a_S \text{ , } t < T \\ C_T = h(X_T) \text{ if } t = T \text{ , } \delta_t = 1 \\ 0 \text{ otherwise} \end{cases}$$

One assume all episodes end at time T, further, if $\delta_t = 0$ for $t < T$, then the trader will choose a_H at each time step in $[t, T]$. Hence, the discounted sum of expected future rewards for a given strategy π is given by

$$V_{\pi,t}(s_t) = E[\sum_{t'=t}^{T} e^{-r(t'-t)} R(s_{t'}, \pi(s_{t'}))|s_t]$$

Thus, the value function V_π satisfies the Bellman equation (see Equation (15.3.13))

$$V_{\pi,t}(s_t) = R(s_t, \pi(s_t)) + e^{-r\Delta t} E[V_{\pi,t+1}(s_{t+1})|s_t]$$

where $\Delta t = (t+1) - t$. Given the structure of the above reward, the Q-value function (see Equation (15.3.20)) satisfies

$$Q_{\pi,t}(s.a) = \begin{cases} e^{-r\Delta t} E[V_{\pi,t+1}(s_{t+1})|s_t] \text{ if } a_t = a_H \\ C_t \text{ if } a_t = a_S \end{cases}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In the special case where the policy π_H is to hold the option till maturity, the value function simplifies to

$$V_{\pi_H,t}(s_t) = e^{-r(T-t)}E[h(X_T)|X_t]$$

which is the option fair price denoted C^F. That is, $V_{\pi_H,t}(s_t) = C_t^F$. From the no-arbitrage condition $C_t = C_t^F$, for all t, and since the underlying is assumed to be a Markov process, the author showed that π_H was an optimal strategy. As a result, the optimal value function V^* satisfies

$$V^*(s_t) = C_t^F \ , \forall s_t$$

Thus, in absence of arbitrage, the optimal value function V^* corresponds to the fair value function C^F. Solving the MDP for V^* solves the derivatives pricing problem. Note, this result is valid even if the market underestimate the fair price, $C_t \leqslant C_t^F$, as π_H would still be an optimal policy. At last, $V^*(s_t) > C_t^F$ provides a measure of mispricing in a flawed market.
This formulation implies that we need to maintain one model per derivative contract, which is impractical. The author proposed to absorb the characteristics of the option into the state, such as, maturity, strike, volatility. It was denoted by ζ in Section (22.1.2) and, for simplicity, we will keep that notation. We get the enhanced state

$$s_t^C = (s_t, \zeta)$$

where it is assumed that the mapping $C \to \zeta$ is bijective. In a discrete state-space, this is a unified notation for accessing the fair price of distinct derivatives. Learning the fair price still takes place separately for each option.

22.3.1.2 The algorithm

In this formulation, the author argues that the dimension of this finite MDP can quickly become so large that the problem becomes computationally unfeasible and consider a continuous state-space framework. In that setting, one tries to directly approximate the optimal value function V^* from a given sample of trajectories from the MDP. This is part of the model-based RL algorithms presented in Section (17.1). The agent can learn a model of how the environment works from its observations and then plan a solution using that model. Once the agent has adequately modelled the environment, it can use a planning algorithm with its learned model to find a policy.
Here, we interpret samples of a high-dimensional discrete MDP as samples from a continuous MDP. The latter is approximated with a simpler discrete MDP. The Kernel-Based RL is chosen.
We assume we have a sample with n transitions $\left(s^{(i)}, \hat{s}^{(i)}, a^{(i)}\right)$ where we go from state $s^{(i)}$ to $\hat{s}^{(i)}$ as a result of action $a^{(i)}$ obtained from the MDP $M = (\mathcal{S}, \mathcal{A}, T, R)$ with continuous state space $\mathcal{S}$, discrete action space $\mathcal{A}$, transition probability $\mathcal{T}$ and reward function R. Following the approach described in Section (17.3), we use the sample to approximate M with a finite MDP $\tilde{M} = (\hat{\mathcal{S}}, \mathcal{A}, \tilde{\mathcal{T}}, \tilde{R})$ where $\hat{\mathcal{S}} = \{\hat{s}^{(i)}\}$ is the set of successor states. Since the market transitions are independent from the chosen action a, we get the simplified expression for $\tilde{T}$ and $\tilde{R}$

$$\tilde{\mathcal{T}}(s, a, \hat{s}^{(i)}) = \tilde{T}(m, \hat{m}^{(i)}) = \frac{1}{Z_m}\phi(\frac{d(m, m^{(i)})}{b})$$

and

$$\tilde{R}(s, a) = \frac{1}{Z_m}\sum_i \phi(\frac{d(m, m^{(i)})}{b})R(s^{(i)}, a)$$

where

$$Z_m = \sum_i \phi(\frac{d(m, m^{(i)})}{b})$$

The state-action value function $\tilde{Q}$ is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\tilde{Q}(s,a) = \tilde{R}(s,a) + e^{-r\Delta t}\sum_i \tilde{T}(m,\hat{m}^{(i)})\tilde{V}(\hat{s}^{(i)}) = \begin{cases} \tilde{R}(s,a) \text{ if } a = a_S \\ e^{-r\Delta t}\sum_i \tilde{T}(m,\hat{m}^{(i)})\tilde{V}(\hat{s}^{(i)}) \text{ otherwise} \end{cases}$$

where $\tilde{V}(s) = \max_a \tilde{Q}(s,a)$. In order to compute V^*, the model-based RL algorithm is as follows:

1. Generate $\tilde{\mathcal{T}}$ by observing enough market transitions.
2. Use a planning algorithm: solve the MDP M using techniques such as value iteration.

In the time-homogeneous Black-Scholes model (see Section (22.1.1.3)), using moneyness η in Equation (22.1.9), the market state simplifies to $m_t = (\eta, \sigma\sqrt{T-t})$. Given the volatility σ we can then evolve the state space.

22.3.2 Hedging options

22.3.2.1 Deterministic BS-model

22.3.2.1.1 Pricing in discrete time We follow the notation on Appendix (28.2) and consider the seller of a European option (put option) with maturity T and payoff $H = h(X(T))$. The replicating portfolio at time t is $V(t)$, the admissible trading strategy is (α, β) and $P(t,T) = e^{-r(t-t)}$ is the zero-coupon bond price. We want to devise a pricing formula that account for intrinsic risk. We choose to minimise the risk of the portfolio by hedging in a sequential decision process. We follow the local risk minimisation approach (see Follmer et al. [1989]). In the mean-variance approach introduced by Markowitz in the portfolio allocation problem (see Appendix (28.1.1)), the risk at time t is given by

$$R_t = \lambda Var_t(V(t))$$

where the parameter λ is a scalar. In that setting, the risk of the portfolio is expressed in terms of its variance. Following the method presented in Appendix (22.5), the risk-averse price for the seller of the option at time $t = 0$ is given by

$$C^{ask}_{t=0}(X,\alpha) = E_0[V_0 + \lambda\sum_{t=0}^{T} P(0,t)Var_t(V(t))|X_0 = X, \alpha_0 = \alpha] \tag{22.3.18}$$

where λ is the risk tolerance (or risk premium) and the second term is the discrete sum of the discounted risk (variance) of a hedge portfolio from all future hedge rebalance periods. This choice of risk premium for option pricing is not unique. We can reformulate this minimisation problem as the maximisation of the objective function

$$V(X,\alpha) \equiv -C^{ask}_{t=0}(X,\alpha)$$

This is a stochastic control problem where the quantity of the asset held α is the control, and the asset X is the state of the system. Assuming normal returns, Halperin [2017] proposed a negative quadratic reward to rewrite the pricing formula in terms of the value function of a finite-horizon MDP problem.

22.3.2.1.2 The value function We need to express the pricing formula in Equation (22.3.18) in terms of the value function to use the techniques developed in reinforcement learning (RL) (see Section (17.1)). That is, we need to identify the reward recovering the option payoff at maturity (see details in Section (17.2.2)). We use the notation in RL where a is the action and s is the state.
A time-homogeneous process X_t, under P, is defined with dynamics

$$dX_t = \sigma d\widehat{W}_t$$

where σ is a constant volatility, $\widehat{W}_t$ is a Brownian motion. Integrating in the range $[t, t+1]$, we get

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\Delta X_t = X_{t+1} - X_t = \sigma \Delta \widehat{W}_t$$

Given $X_0 = logS_0$, the value at time t of the time-dependent stock price is given by the transformation

$$S_t = \varphi(X_t) = e^{X_t + \left(\mu - \frac{\sigma^2}{2}\right)t}$$

where μ is a drift. This is the BS-model with constant volatility and normally distributed returns. In the case of more complex models the function $\varphi(\cdot)$ is more difficult to compute. In the BS-setting, its inverse is given by

$$X_t = logS_t - \left(\mu - \frac{\sigma^2}{2}\right)t$$

The variation in the stock price is given by

$$\Delta S_t = S_{t+1} - S_t = S_t\left(e^{\left(\mu - \frac{\sigma^2}{2}\right)\Delta t + \sigma \Delta \widehat{W}_t} - 1\right)$$

where $E[\Delta S_t | \mathcal{F}_t] = S_t\left(e^{\mu \Delta t} - 1\right)$.

***Remark* 22.3.1** *As a result of the discussion presented in Appendix (22.5), we should not consider the vector of prices $S = (S_0, S_1, ..,)$ but the vector of discounted prices $\overline{S} = (1, \overline{S}_1, ..,)$. Further, we should work under the equivalent probability measure Q where the discounted prices are martingales. In that setting, the discounted stock price is $\overline{S}_t = \frac{S_t}{M(t)}$, where $M(t) = e^{rt}$, with dynamics*

$$\frac{d\overline{S}_t}{\overline{S}_t} = \sigma dW_t$$

where W_t is a Brownian motion under the Q-measure. The value at time t of the discounted stock price is given by the transformation

$$\overline{S}_t = \overline{S}_0 e^{\sigma W_t - \frac{\sigma^2}{2}t}$$

which is a martingale. Note, $E^Q[\Delta \overline{S}_t | \mathcal{F}_t] = 0$.

We denote the action $a_t = a_t(X_t)$ in terms of X and $u_t = u_t(S_t)$ in terms of S. They are related as follows:

$$u_t = a_t\left(\log S_t - \left(\mu - \frac{\sigma^2}{2}\right)t\right)$$

A time dependent policy $\pi(t, X_t)$ is introduced, where $\pi : \{0, ..., T-1\} \times \mathcal{X} \to \mathcal{A}$ is a deterministic policy mapping at time t the state $X_t = x_t$ into the action $a_t \in \mathcal{A}$ given by $a_t = \pi(t, x_t)$.
The pricing formula in Equation (22.3.18) expressed in terms of the spot price S can be rewritten in terms of the time-homogeneous variable X via the function $\varphi(\cdot)$. We introduce the policy π and express the pricing formula in terms of the value function V_π as follows:

$$V_{\pi,t}(X_t) = E_t[-V(t) - \lambda Var(V(t)) - \lambda \sum_{t'=t+1}^{T} P(t, t')Var_{t'}(V(t'))|\mathcal{F}_t]$$

where $V(t)$ is the portfolio value in Equation (28.2.15).
Given the value function with a shifted time argument:

$$V_{\pi,t+1}(X_{t+1}) = E_{t+1}[-V(t+1) - \lambda \sum_{t'=t+1}^{T} P(t+1, t')Var_{t'}(V(t'))|\mathcal{F}_{t+1}]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

we get

$$-\lambda E_{t+1}[\sum_{t'=t+1}^{T} P(t+1,t')Var_{t'}(V(t'))|\mathcal{F}_{t+1}] = V_{\pi,t+1}(X_{t+1}) + E_{t+1}[V(t+1)|\mathcal{F}_{t+1}]$$

We can therefore write

$$\begin{aligned}
-\lambda E_{t+1}[\sum_{t'=t+1}^{T} P(t,t')Var_{t'}(V(t'))] &= -\lambda E_{t+1}[\sum_{t'=t+1}^{T} P(t,t+1)P(t+1,t')Var_{t'}(V(t'))] \\
&= \gamma\Big(V_{\pi,t+1} + E_{t+1}[V(t+1)]\Big)\,, \gamma \equiv P(t,t+1)
\end{aligned}$$

Plugging it back in the above equation, using the recursive Equation (22.5.21) for the portfolio $V(t)$ (and changing notation) and the conditional expectation (see Theorem (24.1.6)), we get the value function

$$\begin{aligned}
V_{\pi,t}(X_t) &= E_t[-V(t) - \lambda Var_t(V(t)) + \gamma\Big(V_{\pi,t+1} + E_{t+1}[V(t+1)]\Big)|\mathcal{F}_t] \\
&= E_t[-\gamma\Big(V(t+1) - a_t\Delta_r S(t)\Big) - \lambda Var_t(V(t)) + \gamma\Big(V_{\pi,t+1} + E_{t+1}[V(t+1)]\Big)|\mathcal{F}_t] \\
&= E_{\pi,t}[r(X_t,a_t,X_{t+1}) + \gamma V_{\pi,t+1}(X_{t+1})]
\end{aligned}$$

where, as in Remark (22.5.1), we use the notation $\Delta_r S(t)$ to distinguish it from the variation $\Delta S(t) = S(t+1) - S(t)$. The one-step time-dependent reward is

$$r_t = r(X_t,a_t,X_{t+1}) = \gamma a_t \Delta_r S_t(X_t,X_{t+1}) - \lambda Var_t(V(t))\,, t = 0,...,T-1$$

where $\Delta_r S_t(X_t,X_{t+1})$ emphasis the fact that it is a function of X_t and X_{t+1}. Using the recursive Equation (22.5.21) and the property of variance (see Appendix (24.1.3)), the conditional variance is

$$\begin{aligned}
Var_t(V(t)) &= \gamma^2 Var_t\big(V(t+1) - a_t\Delta_r S_t\big) \\
&= \gamma^2 Var_t\big(V(t+1)\big) + \gamma^2 a_t^2 Var_t\big(\Delta_r S_t\big) - 2\gamma^2 a_t Cov_t\big(V(t+1),\Delta_r S_t\big) \\
&= \gamma^2\Big(E_t[\big(V(t+1) - \overline{V}(t+1)\big)^2] + a_t^2 E_t[\big(\Delta_r S_t - \overline{\Delta_r S_t}\big)^2] \\
&- 2a_t E_t[\big(V(t+1) - \overline{V}(t+1)\big)\big(\Delta_r S_t - \overline{\Delta_r S_t}\big)]\Big)
\end{aligned}$$

where $\overline{V}(t+1) = \frac{1}{n}\sum_{i=1}^{n} V_i(t+1)$ is the sample mean of all values of $V(t+1)$, and $\overline{\Delta_r S_t}$ is the sample mean of $\Delta_r S_t$. Setting

$$\widehat{V}(t+1) \equiv V(t+1) - \overline{V}(t+1) \text{ and } \Delta_r \widehat{S}_t \equiv \Delta_r S_t - \overline{\Delta_r S_t}$$

the one-step time-dependent reward becomes

$$r_t = \gamma a_t \Delta_r S_t(X_t,X_{t+1}) - \lambda\gamma^2 E_t[\big(\widehat{V}(t+1)\big)^2 - 2a_t\Delta_r\widehat{S}_t\widehat{V}(t+1) + a_t^2\big(\Delta_r\widehat{S}_t\big)^2]$$

At time $t = T$ we get $r_T = -\lambda Var(V(T))$. So the expected reward at time t is quadratic in the action a_t, that is,

$$E_t[r(X_t,a_t,X_{t+1})] = \gamma a_t E_t[\Delta_r S_t] - \lambda\gamma^2 E_t[\big(\widehat{V}(t+1)\big)^2 - 2a_t\Delta_r\widehat{S}_t\widehat{V}(t+1) + a_t^2\big(\Delta_r\widehat{S}_t\big)^2] \tag{22.3.19}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

From Remark (22.5.1), the fair price $\widehat{C}_t$ recover the BS-price only if $\mu = r$, and $E[\Delta_r S_t|\mathcal{F}_t] = 0$. As a result, the above expected reward greatly simplifies.
The optimal policy maximises the value function

$$\pi_t^*(X_t) = \arg\max_{\pi} V_{\pi,t}(X_t)$$

so that the optimal value function satisfies the BOE.
Note, the return r_t corresponds to the expected growth rate for the period between time t and $t+1$ (see Appendix (28.1.1)). That is, it corresponds to the mean-variance criterion for normal returns, where the option is replicated with a quantity of the spot price and some cash. Since, this selection is repeated at each time step, the MDP problem is a multi-period version of the portfolio selection problem.

22.3.2.1.3 The Q-value function We follow the notation in Section (15.3) and let the time-homogeneous process X be the state s and the spot price S be the state s_p.
We can consider the Q-value function with the initial state-action pair (s, a) (see Definition (15.3.6)). Its value at time t is given by

$$Q_{\pi,t}(s,a) = E_t[-V(t,s_t)|s_t = s, a_t = a] - \lambda E_{\pi,t}[\sum_{t'=t}^{T} P(t,t')Var_{t'}(V(t',s_{t'}))|s_t = s, a_t = a]$$

The first expectation only involves averaging over the next time step and as such does not depend on the policy π. Since we can write the value function $V(s)$ in terms of the Q-value function (see Equation (15.3.19)), the Bellman equation for the Q-value function (15.3.20) is

$$Q_{\pi,t}(s,a) = E_t[r_t|(s,a)] + \gamma E_{\pi,t}[V_{\pi,t+1}(s_{t+1})|s]$$

An optimal Q-value function $Q^*_{\pi,t}(s,a)$ is obtained when the Q-value function is evaluated with an optimal policy π_t^* (see Definition (15.3.8)). That is,

$$\pi_t^* = \arg\max_{\pi} Q_{\pi,t}(s,a)$$

The optimal value function and Q-value function are:

$$\begin{aligned} V_t^*(s) &= \max_a Q_t^*(s,a) \\ Q_t^*(s,a) &= E_t[r_t|(s,a)] + \gamma E_t[V_{t+1}^*(s_{t+1})|s] \end{aligned}$$

Plugging the value function in the Q-value function, we get the BOE (see Equation (15.3.24))

$$Q_t^*(s,a) = E_t[r_t + \gamma \max_{a_{t+1}\in\mathcal{A}} Q_{t+1}^*(s_{t+1}, a_{t+1})|s_t = s, a_t = a], t = 0, ..., T-1$$

with terminal condition

$$Q_t^*(s_T, a_T = 0) = -V(T, s_T) - \lambda Var(V(T, s_T))$$

We want a greedy policy π^* that always seeks an action maximising the Q-value function in the current state:

$$\pi_t^*(s_t) = \arg\max_{a_t\in\mathcal{A}} Q_t^*(s_t, a_t)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

22.3.2.2 Learning the optimal policy

Knowing the Q-value function, we can then use some of the methods described in Section (17.1) to solve this optimisation problem. It can be solved with dynamic programming (DP), or, we can also use reinforcement learning (RL). As discussed in Section (17.1), dynamic programming assumes that we know the transition matrix of the environment $P(s^{'}|s,a)$ as well as the expected next step reward. We can then use methods such as Value Iteration and Temporal Difference (TD) learning, among others. In the case where the transition probabilities and the reward function are not known we can directly rely on the data to find an optimal policy and use methods such as Fitted Value Iteration, the Q-learning, or the policy gradients, among others. However, in that case, the pricing formula is not built from the no-arbitrage theory and may not satisfies the Kolmogorov-Compatibility condition (see Appendix (22.6.4)).
We follow the notation in Section (15.3) and let the time-homogeneous process X be the state s and the spot price S be the state s_p.

22.3.2.2.1 Known model Substituting the expected reward in Equation (22.3.19) (quadratic form) into the BOE, we get

$$Q_t^*(X_t,a_t) = \gamma E_t[Q_{t+1}^*(X_{t+1},a_{t+1}^*) + a_t\Delta_r S_t] - \lambda\gamma^2 E_t[\big(\widehat{V}(t+1)\big)^2 - 2a_t\Delta_r\widehat{S}_t\widehat{V}(t+1) + a_t^2\big(\Delta_r\widehat{S}_t\big)^2]$$

which is quadratic in the action variable a_t. In that setting, the optimal action is computed analytically as follows: Differentiating the above equation with respect to a_t and setting the result to zero, we get the optimal action at time t as

$$a_t^*(s_t) = \frac{E_t[\Delta_r\widehat{s}_{p,t}\widehat{V}(t+1) + \frac{1}{2\gamma\lambda}\Delta_r s_{p,t}]}{E_t[\big(\Delta_r\widehat{s}_{p,t}\big)^2]}$$

Following the same approach as in Appendix (22.5.4), we take the limit of the optimal action as $\Delta t \to 0$ by using a Taylor expansion around t. The author got

$$\lim_{\Delta t\to 0} a_t^* = \frac{\partial\widehat{C}_t}{\partial s_p} + \frac{\mu - r}{2\lambda\sigma^2}\frac{1}{s_{p,t}}$$

It is argued by the author that the local risk-minimisation hedge and fair price formulae given in Appendix (22.5) are recovered from the above optimum Q-value function by setting $\mu = r$ and $\lambda = 0$. First, the continuous BS-delta is recovered by setting $\mu = r$. Plugging this result into $Q_t^*(X_t,a_t)$ and taking the limit $\Delta t \to 0$ leads to the fair price $\widehat{C}_t$ in Equation (22.5.24), which yields the BS-equation in the limit $\Delta t \to 0$.

***Remark* 22.3.2** *From Remark (22.5.1), the fair price $\widehat{C}_t$ recover the BS-price only if $\mu = r$, and $E[\Delta_r S_t|\mathcal{F}_t] = 0$. As a result, the above expected reward greatly simplifies and the above optimal action has no-longer the same equation. From Equation (22.6.27), setting $\mu = r$ in the stock dynamics, would result in assuming that the market price of risk λ_p (to distinguish it from the weight) is null.*

***Remark* 22.3.3** *Note, Remark (22.5.6) applies: $\Delta_r s_{p,t}$ is not the right quantity to use and we do not recover the risk-neutral BS-delta by setting the market price of risk to zero, under the historical measure P. Again, to obtain the proper result, we need to work with the discounted prices and the equivalent martingale measure Q.*

***Remark* 22.3.4** *We see that one can not work under the historical measure P and assume non-null interest rates when estimating risk in the theory of portfolio replication. It leads to erroneous results and misleading interpretation.*

Ignoring the above remarks, using this analytical action, the policy optimisation in the Value Iteration method yields the explicit recursive formula:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$Q_t^*(s_t, a_t) = \gamma E_t[Q_{t+1}^*(s_{t+1}, a_{t+1}^*) - \lambda\gamma\big(\widehat{V}(t+1)\big)^2 + \lambda\gamma\big(a_t^*(s_t)\big)^2\big(\Delta_r\widehat{s}_{p,t}\big)^2]\,, t = 0, ..., T-1$$

Thus, in this particular case, no discretisation of the action space $\mathcal{A}$ is required due the quadratic dependence of the Q-value function Q_t on the action a_t at time t. The action in this equation is always just one optimal action. Thus, the fair ask option price is $C_t^{QLBS} = -Q_t(s_t, a_t^*)$. The whole calculation is semi-analytic.
In the case of continuous-space scenarios with MC simulations, the optimal hedge and Q-function are expanded over some set of basis functions and the results are expressed in terms of the expansion coefficients. Rather than using a lattice, which applies to a finite-state version of the model, the MC is more general as it average over all scenarios from time $[t, t+1]$ simultaneously. Therefore, there is no need in explicit conditioning on s_t at time t. Learning optimal actions for all states simultaneously means learning a policy (see Remark (15.3.5)), and as such it applies to RL for unknown model.

22.3.2.2.2 Unknown model In the cases where the transition probabilities and rewards are not known, the Bellman optimality equation (BOE) is solved recursively by using samples. Since the reward function is quadratic, the author obtained semi-analytical solutions by using the Q-learning method. However, to guarantee convergence to the optimal hedge we need enough training data.

Given some historical data and no access to a real-time environment, the author considered a batch-mode learning with a set of N trajectories and information set $\mathcal{F}_t = \{\mathcal{F}_t^{(n)}; n = 1, ..., N\}$, which contains the asset (or state) s_t, the hedge position a_t, the reward r_t and the next-time state s_{t+1}. That is,

$$\mathcal{F}_t^{(n)} = \{\big(s_t^{(n)}, a_t^{(n)}, r_t^{(n)}, s_{t+1}^{(n)}\big); t = 0, .., T-1\}$$

The dynamics are assumed to be Markov so that we have $N \times T$ single-step transitions $P(s^{'}|s, a)$ given.

Assumption 5 *Either the data is simulated, or some historical stock price is combined with artificial data to track the performance of a hypothetical stock and cash replicating portfolio for a given option. Neither the asset dynamics nor the true reward distribution are known. We only know a set of one-step transition.*

The value function can be decomposed into immediate reward plus discounted value of successor states (see Equation (15.3.13)). The Bellman equation at time t is

$$V_{\pi,t}(s_t) = E_{\pi,t}[r_t + \gamma V_{\pi,t+1}(s_{t+1})]$$

where $r_t = r(s_t, a_t, s_{t+1})$ is a one-step time-dependent random reward. In that setting, the risk-adjusted portfolio return, or, the expected growth rate, is given by

$$r_t = \gamma a_t \Delta_r s_{p,t} - \lambda Var_t(V(t))$$

where $\Delta_r s_{p,t} = s_{p,t+1} - e^{-r\Delta t} s_{p,t}$.
The author used the Fitted Q-Iteration method to solve this problem, which combines the Q-learning method with function approximations to deal with large state spaces (see Section (17.1.3)).

In the Q-learning algorithm, given the observation $\big(s_t^{(n)}, a_t^{(n)}, r_t^{(n)}, s_{t+1}^{(n)}\big)$, we update the action-value function for a given state-action node $\big(s_t^{(n)}, a_t^{(n)}\big)$. However, convergence of the Q-learning method (see Theorem (17.1.1)) requires an exhaustive sampling of the state-action pairs (s, a). Even if the training data (in the form of tuples) comes from an option trading desk it might not be sufficient to obtain convergence (asymptotically large data set) and one might have to artificially simulate some data. Note, the actions can be completely random and the Q-learner will still learn the correct price and hedge provided that there is sufficient data.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Further, since optimal hedges are obtained using cross-sectional information across all MC paths, such information could be masked in the updating process.

In the Fitted Q-Iteration algorithm the learning is in a batch-mode. We have access to N simulated (or real) paths for the state variable. While neither the transition probabilities nor the reward functions are known, we have two extra pieces of information per each observed state vector (compared to the Q-learning): the action a_t and the reward r_t. That is, we do not know the functions π_t^* and r_t, but have samples from these functions in different states. So the state variable values and rewards at both time t and $t+1$ for all MC paths are used simultaneously the find the optimal action and optimal Q-function at time t, for all states and actions.
Practically, the Monte Carlo method is used to simulate all paths for the replicating portfolio simultaneously, and the average is done simultaneously over all scenarios between time t and $t+1$. As a result, there is no need to explicitly condition on the state s_t at time t. Thus, the update of the Q-values and actions a_t^* at all points on the grid is done simultaneously by looking at each time t at a snapshot of all MC paths. This is because in the batch-mode learning all the data is already there.

More sophisticated learning algorithms could be used such as deep RL, policy gradients methods, and actor-critic algorithms (see Section (17.1)).

Halperin [2018] provided a numerical example using simulated stock price histories S_t with initial stock price $S_0 = 100$ and constant volatility $\sigma = 0.15$. A European put option is considered with maturity is $T = 1$ year, strike $K = 100$, hedging twice a week, and $50,000$ MC scenarios for a path of the stock. Results are reported with two MC runs. Thus, the framework is a discrete BS-model which is does not correspond to market conditions. Comparisons between the continuous and discrete BS-model are produced. In the case of the on-policy learning, the optimal actions and rewards obtained with DP are used as inputs to the FQI algorithm in addition to the paths of the asset. In the case of the off-policy learning, the optimal hedges obtained with DP are multiplied by a random uniform number. Five different scenarios of sub-optimal actions are considered. In reality, as the spot moves the volatility will change impacting the hedging portfolio.

***Remark* 22.3.5** *In most cases, traders do not hold an option till maturity, so that a trading desk do not have the hedge position over the life of the option. In order to generate artificial data for tracking the performance of a hypothetical stock and cash replicating portfolio for a given option, one need a model to devise an arbitrage-free implied volatility for the given spot level, maturity and strike.*

22.3.2.3 Non-normal rate model

There are two parts to the pricing and hedging of options in a discrete-time framework described in Section (22.3.2.1):

1. Pricing and hedging in incomplete market: one must select a measure of risk and devise the associated pricing formula.

2. Formulating the Bellman equation: one has to derive the Bellman equation associated with the pricing formula.

Both parts are model dependent. The simple fact of assuming that local rewards are quadratic a-la Markowitz, already implies that market returns are normally distributed. It explains why the expected reward is quadratic with respect to the action, leading to computational simplification. One can not states that the FQI is model-free and does not require the need of the volatility surface or any jump-diffusion models, since we considered constant volatility σ for the dynamics of the spot price and assumed that the risk was defined in terms of the portfolio variance $f(\sigma)$ (see Equation (22.1.6)). Should we consider discontinuous processes, the quadratic variation would be very different (see Appendix (25.1.1.3)), leading to very different pricing and hedging formulae.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

22.3.2.3.1 The assumption of normal returns We also distinguish two parts to the pricing and hedging of the deterministic BS-model presented in Section (22.3.2.1):

1. The hedging: minimising the portfolio variance across all simulated MC paths.
2. The pricing: adding the cumulative expected discounted variance of the hedge portfolio along all time steps with a risk-aversion parameter λ. Then minimise this option price.

In both cases market risk is expressed in terms of portfolio variance, and in both cases we want to minimise that risk. Hence, hedging and pricing is consistent, so that we get a single computable formula for pricing and hedging options. However, the variance is only a measure of risk in the special case of second-order processes among which are the Gaussian processes (see Appendix (28.1.2)). In the case of non-normal returns, higher moments need to be taken into consideration. In some models the variance is not even defined.
The formulation of the MDP problem is clearly model dependent because:

1. in order to transform the pricing formula into the Bellman equation we assumed the underlying returns to be normally distributed, allowing for an easy transformation function φ between the non-stationary spot price S and the time-homogeneous variable X.
2. the pricing and hedging formula have been derived to minimise variance, which is consistent with normal market returns.

As a result, the computation of the reward in the MDP problem is quadratic in the action, leading to analytical solution to the optimal policy. This is no-longer the case if we consider non-normal returns.

22.3.2.3.2 Beyond normal returns The method of pricing and hedging options described in Section (22.3.2.1) corresponds to the utility-based approach, where the quadratic utility function has been chosen. It is generally used for its excellent tractability properties, but a risk averse investor with a quadratic utility will increase the percentage of his wealth invested in risky assets as his wealth increases.
The introduction of portfolio variance as a measure of risk in both the pricing and hedging formulae is consistent with the mean-variance criterion in the CMAP approach of portfolio selection (see Appendix (28.1.1.1)) when returns are normally distributed.
It comes from the fact that an exponential utility $U(x) = -e^{-\gamma x}$ can be approximated to give a quadratic certain equivalent (CE) of the portfolio. Taking the limit $\gamma \to 0$ and making a correction via an expansion in powers of γ, an approximate relation between the parameter λ of the quadratic minimisation and the parameter γ of the exponential utility is given by $\lambda \approx \frac{1}{2}\gamma$ (see Appendix (28.1.1.1)).

The optimal hedge α^* corresponds to solving the optimisation problem in Equation (28.1.1), that is, minimising the portfolio variance when the drift μ is a constant. However, we are implicitly assuming that returns are normally distributed. When returns are non-normal, the certain equivalent (CE) of the portfolio also depends on skewness and kurtosis.

22.3.2.3.3 The modified risk-averse price As discussed in Appendix (28.1.2), variance is a flawed measure of risk when returns are non-normal. The agent (or decision maker) should prefer high average returns, lower variance or standard deviation, positive skewness, and lower kurtosis. Thus, alternative measures of risk should be considered. That is, a risk averse investor having an exponential utility has aversion to risk associated with increasing variance, negative skewness, and increasing kurtosis. Accounting for these risk, the optimal hedge at time t should be

$$\begin{aligned}
\alpha_t^* &= \arg\min_{\alpha}\big(Var_t - S_t + K_t\big) \\
&= \arg\min_{\alpha}\Big[Var_t\big(V(t+1) - \alpha(t)\Delta X(t)\big) - S_t\big(V(t+1) - \alpha(t)\Delta X(t)\big) + K_t\big(V(t+1) - \alpha(t)\Delta X(t)\big)\Big] \\
&\quad , t = T-1, ..., 0
\end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $Var_t = Var_t(V(t))$ is the variance of the portfolio, $S_t = S_t(V(t))$ is the skewness, and $K_t = K_t(V(t))$ is the kurtosis. Obviously, this is no-longer a quadratic minimisation problem.
Similarly, we can build a modified risk-averse price by adding the cumulative expected discounted variance of the hedge portfolio, subtracting the cumulative expected discounted skewness of the hedge portfolio, and adding the cumulative expected discounted kurtosis of the hedge portfolio. In that case, the modified pricing formula becomes

$$\tilde{C}_0^{ask}(X,\alpha) = E_0[V(0)+\lambda_V \sum_{t=0}^{T} e^{-rt} Var_t(V(t)) - \lambda_S \sum_{t=0}^{T} e^{-rt} S_t(V(t)) + \lambda_K \sum_{t=0}^{T} e^{-rt} K_t(V(t)) | X(0) = X, \alpha_0 = \alpha]$$

where λ_V is the risk aversion parameter for the variance, λ_S is the risk aversion parameter for the skewness, and λ_K is the risk aversion parameter for the kurtosis.

22.3.2.3.4 The modified value function Changing notation, we can reformulate this problem as the maximisation of the objective function

$$V(S,a) \equiv -\tilde{C}_{t=0}^{ask}(S,a)$$

We need to define a dynamics of the process X as well as a transformation function to express the stock price S in terms of the process X. We will not describe this now.
The modified pricing formula can be rewritten in terms of the value function as follows:

$$V_{\pi,t}(X_t) = E_t[-V(t) - \lambda_V Var_t + \lambda_S S_t - \lambda_K K_t - \lambda_V \sum_{t'=t+1}^{T} P(t,t') Var_{t'} + \lambda_S \sum_{t'=t+1}^{T} P(t,t') S_{t'} - \lambda_K \sum_{t'=t+1}^{T} P(t,t') K_{t'} | \mathcal{F}_t]$$

Given the value function with a shifted time argument:

$$V_{\pi,t+1}(X_{t+1}) = E_{t+1}[-V(t+1) - \lambda_V \sum_{t'=t+1}^{T} P(t+1,t') Var_{t'} + \lambda_S \sum_{t'=t+1}^{T} P(t+1,t') S_{t'} - \lambda_K \sum_{t'=t+1}^{T} P(t+1,t') K_{t'} | \mathcal{F}_{t+1}]$$

we get

$$\begin{aligned} &-\lambda_V E_{t+1}[\sum_{t'=t+1}^{T} P(t+1,t') Var_{t'} | \mathcal{F}_{t+1}] + \lambda_S E_{t+1}[\sum_{t'=t+1}^{T} P(t+1,t') S_{t'} | \mathcal{F}_{t+1}] \\ &-\lambda_K E_{t+1}[\sum_{t'=t+1}^{T} P(t+1,t') K_{t'} | \mathcal{F}_{t+1}] = V_{\pi,t+1}(X_{t+1}) + E_{t+1}[V(t+1) | \mathcal{F}_{t+1}] \end{aligned}$$

We can therefore write

$$\begin{aligned} & -\lambda_V E_{t+1}[\sum_{t'=t+1}^{T} P(t,t') Var_{t'}] + \lambda_S E_{t+1}[\sum_{t'=t+1}^{T} P(t,t') S_{t'}] - \lambda_K E_{t+1}[\sum_{t'=t+1}^{T} P(t,t') K_{t'}] \\ = & -\lambda_V E_{t+1}[\sum_{t'=t+1}^{T} P(t,t+1) P(t+1,t') Var_{t'}] + \lambda_S E_{t+1}[\sum_{t'=t+1}^{T} P(t,t+1) P(t+1,t') S_{t'}] \\ & -\lambda_K E_{t+1}[\sum_{t'=t+1}^{T} P(t,t+1) P(t+1,t') K_{t'}] \\ = & \gamma\Big(V_{\pi,t+1} + E_{t+1}[V(t+1)]\Big), \gamma \equiv P(t,t+1) \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Plugging it back in the above equation, and using the recursive Equation (22.5.21) (and changing notation), we get the value function

$$\begin{aligned} V_{\pi,t}(X_t) &= E_t[-V(t) - \lambda_V Var_t + \lambda_S S_t - \lambda_K K_t + \gamma\Big(V_{\pi,t+1} + E_{t+1}[V(t+1)]\Big)|\mathcal{F}_t] \\ &= E_t[-\gamma\Big(V(t+1) - a_t \Delta S(t)\Big) - \lambda_V Var_t + \lambda_S S_t - \lambda_K K_t + \gamma\Big(V_{\pi,t+1} + E_{t+1}[V(t+1)]\Big)|\mathcal{F}_t] \\ &= E_{\pi,t}[r(X_t, a_t, X_{t+1}) + \gamma V_{\pi,t+1}(X_{t+1})] \end{aligned}$$

where the one-step time-dependent reward is

$$r_t = r(X_t, a_t, X_{t+1}) = \gamma a_t \Delta S_t(X_t, X_{t+1}) - \lambda_V Var_t + \lambda_S S_t - \lambda_K K_t \text{ , } t = 0, ..., T-1$$

Clearly, the expected reward at time t is no-longer quadratic in the action a_t, but it is raised to the power four. It corresponds to the portfolio selection problem between time t and $t+1$ in the case of non-normal market rates.
We can then derive the Q-value function and use some of the methods described in Section (17.1) to solve this optimisation problem. However, we can no-longer rely on analytical solutions.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

22.4 Appendices

22.5 Local risk minimisation

We briefly describe a local risk minimisation approach presented by Grau [2007] and modified by Halperin [2017] to obtain an optimal strategy and a pricing formula by minimising portfolio variance as a measure of risk. This method is consistent with the assumption of normality for the market returns.
We show some of the inconsistencies of this approach leading to erroneous results and misleading interpretation, and propose an alternative solution.
We use the same assumptions and notation as in Appendix (28.2.4). That is, the process X is a continuous semimartingale and the payoff H is in the Hilbert space L_2. The value of a portfolio at time t satisfies Equation (28.2.15), that is,

$$V(t) = \alpha(t)X(t) + \beta(t)M(t)\,,\, 0 \leqslant t \leqslant T$$

where α is the the quantity of risky asset hold and β is the quantity of cash.

22.5.1 Portfolio evaluation

We consider the seller of an option with maturity T and terminal payoff H. Starting from maturity T, we want to find the amount needed to be held in the bank account at the previous times $t < T$.

22.5.1.1 Description

Given a self-financing strategy (see Appendix (28.2.2.3)), the following relation ensures conservation of the portfolio value by re-hedging at time $t+1$:

$$\alpha(t)X(t+1) + e^{r\Delta t}\beta(t) = \alpha(t+1)X(t+1) + \beta(t+1) \tag{22.5.20}$$

We can therefore get a recursive relation used to compute the amount of money to keep in the bank account to hedge the option at any time $t < T$:

$$\beta(t) = e^{-r\Delta t}\Big(\beta(t+1) + \big(\alpha(t+1) - \alpha(t)\big)X(t+1)\Big)\,,\, t = T-1, ..., 0$$

Combining this with the above portfolio value, we get a recursive relation for the portfolio $V(t)$ in terms of its values at later times, which we solve backward in time with terminal condition H. We get the hedged portfolio

$$V(t) = e^{-r\Delta t}\Big(V(t+1) - \alpha(t)\Delta_r X(t)\Big)\,,\, \Delta_r X(t) = X(t+1) - e^{r\Delta t}X(t)\,,\, t = T-1, ..., 0 \tag{22.5.21}$$

***Remark* 22.5.1** *We use the notation $\Delta_r X(t)$ to distinguish it from the variation $\Delta X(t) = X(t+1) - X(t)$.*

In that setting, both $\beta(t)$ and $V(t)$ are not measurable at any time $t < T$ since they depend on the future. All uncertainties in $V(t)$ are due to the uncertainty regarding the amount $\beta(t)$ to be held at the bank at time t to cover the payoff H at maturity T. At time $t = 0$, both $\beta(0)$ and $V(0)$ are random quantities with some distributions.
In that setting, the computation of the hedge portfolio value is done path-wise. Given some hedging strategies $\{\alpha(t); t = 0, ..., T\}$ we can infer these distributions by using Monte Carlo simulations. We can simulate N paths for X and go backward on each path to evaluate $V(t)$ (see Grau [2007]). That is,

1. In the forward pass we simulate the process $X(1) \to X(2) \to \cdots \to X(N)$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. In the backward pass we use the recursion in Equation (22.5.21) that takes a given hedge strategy $\{\alpha(t); t = 0, ..., T\}$ and back-propagate future uncertainties till today via the self-financing constraint in Equation (22.5.20).

We can also estimate $V(0)$ from historical data for X together with a pre-determined hedging strategy and a terminal condition.

22.5.1.2 Problems and solutions

22.5.1.2.1 An overview We have introduced in Remark (22.5.1) a new notation $\Delta_r X(t)$ to distinguish it from the variation $\Delta X(t)$. It reflects the fact that the recursive relation in Equation (22.5.21) does not correspond to the representation of portfolios in financial models both in discrete and continuous time.
For instance, assuming the dynamics on the asset $X(t)$ to satisfy Equation (22.6.27), we get

$$X(t+1) = X(t)e^{(\mu - \frac{1}{2}\sigma^2)\Delta t + \sigma \Delta \widehat{W}_t}$$

where $\mu = r + \sigma\lambda$ and λ is the market price of risk. Thus,

$$\Delta_r X(t) = X(t)e^{r\Delta t}\left(e^{(\mu - r - \frac{1}{2}\sigma^2)\Delta t + \sigma \Delta \widehat{W}_t} - 1\right) \text{ and } \Delta X(t) = X(t)\left(e^{(\mu - \frac{1}{2}\sigma^2)\Delta t + \sigma \Delta \widehat{W}_t} - 1\right)$$

such that

$$E[\Delta_r X(t)|\mathcal{F}_t] = X(t)e^{r\Delta t}\left(e^{(\mu - r)\Delta t} - 1\right) \text{ and } E[\Delta X(t)|\mathcal{F}_t] = X(t)\left(e^{\mu \Delta t} - 1\right) \tag{22.5.22}$$

Taking the limit $\Delta t \to 0$, we get

$$\lim_{\Delta t \to 0} E[\Delta_r X(t)|\mathcal{F}_t] = X(t)(\mu - r)\Delta t \text{ and } \lim_{\Delta t \to 0} E[\Delta X(t)|\mathcal{F}_t] = X(t)\mu\Delta t$$

From Remark (28.2.6), under the historical measure P, only in the case $\mu = r$ do we recover the BS-PDE in Equation (28.2.13). In that case, $E[\Delta_r X(t)|\mathcal{F}_t] = 0$. Another way of saying this is as follows: Expressing the asset under the risk-neutral measure $\mathbb{Q}$, we get

$$X(t+1) = X(t)e^{(r - \frac{1}{2}\sigma^2)\Delta t + \sigma \Delta W_t}$$

Thus,

$$\Delta_r X(t) = X(t)e^{r\Delta t}\left(e^{-\frac{1}{2}\sigma^2 \Delta t + \sigma \Delta W_t} - 1\right) \text{ and } \Delta X(t) = X(t)\left(e^{(r - \frac{1}{2}\sigma^2)\Delta t + \sigma \Delta W_t} - 1\right)$$

such that

$$E^Q[\Delta_r X(t)|\mathcal{F}_t] = 0 \text{ and } E^Q[\Delta X(t)|\mathcal{F}_t] = X(t)\left(e^{r\Delta t} - 1\right)$$

As a result, one can not apply a control on the quantity of stock to hold, under the risk-neutral measure, using a backward portfolio equation.

As discussed in Appendix (28.2.1), when the interest rate is non-null we need to take the money market as numeraire and work under an equivalent martingale measure where discounted prices are martingale. For instance, in the discrete model, the recursive relation for the discounted portfolio is given in Equation (28.2.7).
In that setting, we work with the discounted portfolio $\overline{V}(t) = \frac{V(t)}{M(t)}$ and the discounted asset $\overline{X}(t) = \frac{X(t)}{M(t)}$, where the money market is $M(t) = e^{rt}$. We consider the self-financing condition for discounted prices given in Equation (28.2.11) and get, at time $t+1$, the relation

$$\alpha(t)\overline{X}(t+1) + \beta(t) = \alpha(t+1)\overline{X}(t+1) + \beta(t+1)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We can therefore get a recursive relation used to compute the amount of money to keep in the bank account to hedge the option at any time $t < T$:

$$\beta(t) = \beta(t+1) + \big(\alpha(t+1) - \alpha(t)\big)\overline{X}(t+1)\,, t = T-1, ..., 0$$

Combining this with the discounted portfolio value

$$\overline{V}(t) = \alpha(t)\overline{X}(t) + \beta(t)$$

we get the recursive relation

$$\overline{V}(t) = \overline{V}(t+1) - \alpha(t)\Delta\overline{X}(t)\,, t = T-1, ..., 0$$

22.5.1.2.2 The recursive portfolio equation We follow the notation in Appendix (28.2.3) and consider the portfolio at time t_k given by

$$V_{t_k}(\theta) = \alpha_k S_1(t_k) + \beta_k S_0(t_k)$$

which is $\mathcal{F}_{t_k}$-measurable.
In the BS-framework, the quantity of cash is $\beta_k = \frac{Z_{t_k}}{S_0(t_k)}$ (see Theorem (28.2.2)), so that the portfolio becomes

$$V_{t_k}(\theta) = \alpha_k S_1(t_k) + Z_{t_k}$$

Recall, the cash asset S_0 describes the value of \$1 daily compounded (rolled over $]t_k, t_{k+1}]$) at the bank. We assume the cash is compounded with the gross rate in Equation (22.1.1). The self-financing condition, defined in Equation (28.2.9), becomes

$$\alpha_k S_1(t_{k+1}) + \beta_k S_0(t_{k+1}) = \alpha_{k+1} S_1(t_{k+1}) + \beta_{k+1} S_0(t_{k+1})$$

In our setting, we have

$$\beta_k S_0(t_{k+1}) = Z_{t_k}\frac{S_0(t_{k+1})}{S_0(t_{k+})} = Z_{t_k}e^{r\Delta t}$$

so that the above equation simplifies to

$$\alpha_k S_1(t_{k+1}) + Z_{t_k}e^{r\Delta t} = \alpha_{k+1} S_1(t_{k+1}) + Z_{t_{k+1}}$$

From Remark (28.2.2), for all t in the interval $]t_k, t_{k+1}]$ then $\beta(t) = \beta_{t_{k+1}} = n^0(k)$. Thus, Z_{t_k} on the left-hand-side (LHS) is $\mathcal{F}_{t_k}$-measurable. However, on the right-hand-side (RHS), the quantities α_{k+1} and β_{k+1} are $\mathcal{F}_{t_{k+1}}$-measurable. Having invested $\alpha_k S_1(t_k)$ just after t_k and put Z_{t_k} at the bank, the wealth of the agent at time t_{k+1} is the LHS of the above equation. This is the wealth the agent reinvest just after time t_{k+1}. Thus, we should not express β_{t_k} in terms of $\beta_{t_{k+1}}$.
Nonetheless, computing for Z_{t_k}, we get

$$Z_{t_k} = e^{-r\Delta t}(\alpha_{k+1} - \alpha_k)S_1(t_{k+1}) + e^{-r\Delta t}Z_{t_{k+1}}$$

which is no-longer an $\mathcal{F}_{t_k}$-measurable variable as it depends on α_{k+1} and $S_1(t_{k+1})$.
The portfolio $V_{t_k}(\theta)$ is $\mathcal{F}_{t_k}$-measurable. Nonetheless, replacing the above Z_{t_k} in the portfolio, we get

$$V_{t_k}(\theta) = e^{-r\Delta t}\Big[V_{t_{k+1}}(\theta) - \alpha_k\big(S_1(t_{k+1}) - S_1(t_k)e^{r\Delta t}\big)\Big]$$

which is no-longer an $\mathcal{F}_{t_k}$-measurable.
However, when considering the assets' variation between two trading dates, the self-financing condition in Equation (28.2.10) becomes

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$V_{t_{k+1}}(\theta) = V_{t_k}(\theta) + \alpha_k \Delta_k S_1 + \beta_k \Delta_k S_0$$

where $\Delta_k S_0 = S_0(t_{k+1}) - S_0(t_k)$ and $\Delta_k S_1 = S_1(t_{k+1}) - S_1(t_k)$. We have $\Delta_k S_0 = S_0(t_k)\big(e^{r\Delta t} - 1\big)$ so that $\beta_k \Delta_k S_0 = Z_{t_k}\big(e^{r\Delta t} - 1\big)$. Replacing in the above equation, we get

$$V_{t_{k+1}}(\theta) = V_{t_k}(\theta) + \alpha_k \Delta_k S_1 + Z_{t_k}\big(e^{r\Delta t} - 1\big)$$

We recover the fact that the variations of a self-financing portfolio are only due to the variations of the assets.

22.5.2 Optimal hedging

22.5.2.1 Description

In order to set the price of the option at inception, the seller first need to select a hedging strategy α to be used in the future. In that setting, the theory relies on the principle of hedging first and pricing second. Thus, we need to learn the optimal strategy α^* over all states of the world. That is, optimal hedges must be computed backward in time, starting from $t = T$, and using a cross-sectional analysis on all paths.
Following a similar approach to the pricing of Amrican options (see Longstaff et al. [2001]), Halperin [2017] computed the optimal hedge α_t^* at time t by minimising the portfolio variance across all simulated MC paths, when conditioned on the currently available cross-sectional information $\mathcal{F}_t$. Using Equation (22.5.21) for the recursive portfolio, we get the optimal hedge at time t as

$$\begin{aligned}\alpha_t^* &= \arg\min_{\alpha} Var_t(V(t)) \\ &= \arg\min_{\alpha} Var_t\big(V(t+1) - \alpha(t)\Delta_r X(t)\big)\ , t = T-1, ..., 0\end{aligned}$$

where $Var_t(X) = Var(X|\mathcal{F}_t)$. The optimal hedge should minimise the cost of hedge capital at each time step. Using the results in Appendix (24.1.3), we rewrite the variance as follows:

$$Var_t\big(V(t+1) - \alpha(t)\Delta_r X(t)\big) = Var_t\big(V(t+1)\big) + \alpha^2(t) Var_t\big(\Delta_r X(t)\big) - 2\alpha(t) Cov_t\big(V(t+1), \Delta_r X(t)\big)$$

where $Cov_t(X,Y) = Cov(X,Y|\mathcal{F}_t)$. Differentiating the above equation with respect to α and setting the result to zero, we get the optimal hedge at time t as

$$\alpha_t^* = \frac{Cov_t(V(t+1), \Delta_r X(t))}{Var(\Delta_r X(t))}\ , t = T-1, ..., 0 \tag{22.5.23}$$

which corresponds to Equation (28.2.16).
These terms are just one-step ahead conditional expectations:

- In the case of discrete state space they are finite sums involving the transition probabilities of a MDP model. That is, the finite sums are over all feature discrete states reachable in one step from a given discrete state.
- In the case of continuous state space they can be computed with a Monte Carlo engine by using expansions in basis functions (see Section (17.1.3.7)).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

22.5.2.2 Problems and solutions

As discussed in Appendix (28.2.1), in the discrete case, if we account for the cost process, the recursive equation for a portfolio is given by Equation (28.2.6). In a complete market there is a measure Q equivalent to P with one self-financing strategy replicating a payoff H at maturity. In that case, the cost process is a constant and H is replicated exactly. The optimum strategy is given in Equation (28.2.16) In an incomplete market, for a given martingale measure, we need to find a strategy minimising the risk in some sense. Follmer et al. [1986] and Schweizer [1988] introduced the quadratic risk of the cost process. Details are given in Appendix (28.2.4). A weak formulation consists in minimising the local risk presented in Definition (28.2.11). Thus, we can view the approach presented by Halperin [2017] as assuming complete market and minimising the local risk. The optimum strategy is given in Equation (28.2.8). Note, this solution is expressed with respect to ΔX and not $\Delta_r X$. Again, we can recover the result by working with the discounted prices $\overline{S}$ under the equivalent martingale measure Q.

22.5.3 The risk-averse price

Since the portfolio $V(t)$ satisfies the backward Equation (22.5.21), the price of a fair option $\widehat{C}_t$, at time t, is the expected value of the hedged portfolio given by

$$\widehat{C}_t = E[V(t)|\mathcal{F}_t]$$

Using Equation (22.5.21) for the recursive portfolio and conditional expectation, we get

$$\begin{aligned}\widehat{C}_t &= E[e^{-r\Delta t}V(t+1)|\mathcal{F}_t] - \alpha(t)e^{-r\Delta t}E[\Delta_r X(t)|\mathcal{F}_t] \\ &= E[e^{-r\Delta t}E[V(t+1)|\mathcal{F}_{t+1}]|\mathcal{F}_t] - \alpha(t)e^{-r\Delta t}E[\Delta_r X(t)|\mathcal{F}_t] \\ &= E[e^{-r\Delta t}\widehat{C}_{t+1}|\mathcal{F}_t] - \alpha(t)e^{-r\Delta t}E[\Delta_r X(t)|\mathcal{F}_t]\,, t = T-1,...,0 \end{aligned} \tag{22.5.24}$$

From Remark (22.5.1), to recover the BS-PDE we must set $\mu = r$, and $E[\Delta_r X(t)|\mathcal{F}_t] = 0$. We can deduce that

$$\widehat{C}_t = E^Q[e^{-r\Delta t}\widehat{C}_{t+1}|\mathcal{F}_t] \tag{22.5.25}$$

Following the same approach as in the above section, we can express the optimal hedge in Equation (22.5.23) in terms of $\widehat{C}_{t+1}$ instead of $V(t+1)$, getting

$$\alpha_t^* = \frac{Cov_t(\widehat{C}_{t+1}, \Delta_r X(t))}{Var(\Delta_r X(t))}\,, t = T-1,...,0 \tag{22.5.26}$$

Substituting the optimal hedge in the pricing equation above, we get

$$\widehat{C}_t = e^{-r\Delta t}E^{\widehat{Q}}[V(t)|\mathcal{F}_t]\,, t = T-1,...,0$$

where $\widehat{Q}$ is a signed measure with the following transition probabilities:

$$\tilde{q}(X(t+1)|X(t)) = p(X(t+1)|X(t))\Big(1 - \frac{\big(\Delta_r X(t) - E_t[\Delta_r X(t)]\big)E_t[\Delta_r X(t)]}{Var_t(\Delta_r X(t))}\Big)$$

where $p(X(t+1)|X(t))$ are transition probabilities under the historical measure $\mathbb{P}$. The measure is signed because the transition probabilities can become negative for large moves $\Delta_r X(t)$. It is a property of quadratic risk minimisation schemes.

***Remark* 22.5.2** *As such this pricing formula does not rely on the no-arbitrage principle. The author argue that if the data respect the no-arbitrage principle, the model will also respect it because it is directly build from data. This is very difficult to prove in practice*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, the seller has to compensate for the risk of exhausting the bank account $\beta(t)$ before reaching maturity and having to inject extra cash in the system.

***Remark* 22.5.3** *This statement stresses that there is a cost process involved with an expected value not equal to a constant.*

Potters et al. [2001] suggested adding the cumulative expected discounted variance of the hedge portfolio along all time steps with a risk-aversion parameter λ. The pricing formula becomes

$$C_0^{ask}(X,\alpha) = E_0[V(0) + \lambda \sum_{t=0}^{T} e^{-rt} Var_t(V(t))|X(0) = X, \alpha_0 = \alpha]$$

***Remark* 22.5.4** *This pricing formula can be made non-negative by choosing a proper level for the risk aversion parameter λ.*

22.5.4 The BS limit

We let C_t be the BS-price given in Equation (22.1.7) under the risk-neutral measure Q. The classical pricing formulation in Equation (22.1.3) is the result of a forward diffusion where $h(X(T))$ is a random variable, and its expectation at time t, denoted C_t, is a known value. However, in the case of the fair price $\widehat{C}_t$ in Equation (22.5.24), the replicating portfolio $V(T) = h(X(T))$ is a known value at maturity T, and $V(t)$ is a random value at time t. Nonetheless, the fair price $\widehat{C}_t$ is known at time t and must be equal to the option price C_t. That is,

$$C_t \equiv \widehat{C}_t$$

From Remark (22.5.1), this only occurs if $\mu = r$, and $E[\Delta_r X(t)|\mathcal{F}_t] = 0$. Further, the decomposition of C_t uses a forward operator, while the decomposition $\widehat{C}_t$ uses a backward operator.
Halperin [2017] considered a Taylor expansion (see Appendix (22.6.2.1)) coupled with a limit argument to prove that the fair option price $\widehat{C}_t$ was equivalent to the BS-price under the risk-neutral measure. The author assumed the backward price $\widehat{C}_{t+1}$ to be close to the classical forward price C_t, and Taylor expanded the former with respect to the latter. Assuming the dynamics on the asset $X(t)$ to satisfy Equation (22.6.27), and taking the second-order Taylor expansion on the fair price $\widehat{C}$, he got

$$\begin{aligned}\widehat{C}_{t+1} &= C_t + \frac{\partial C_t}{\partial t}dt + \frac{\partial C_t}{\partial X}dX(t) + \frac{\partial^2 C_t}{\partial X^2}(dX(t))^2 + \cdots \\ &= C_t + \frac{\partial C_t}{\partial t}dt + \frac{\partial C_t}{\partial X}X(t)\big(\mu dt + \sigma dW_t\big) + \cdots\end{aligned}$$

where $dX(t)$ is the limit of $\Delta X(t)$, which is the variation on X (see Remark (22.5.1)). The right hand side (RHS) of the above expression corresponds to the expansion in Equation (22.6.34), which is a forward equation. On the other hand, the left hand side (LHS) is a backward price.

***Remark* 22.5.5** *As discussed in Appendix (22.6.2.1), the Taylor expansion for a function $f(t,x)$ is given in Equation (22.6.32). The expansion is performed around a point, say (a,b), so that we get $f(a,b)$. This is the same function f. Here, the backward price $\widehat{C}_{t+1}$ is expanded around the forward price C_t, which is not the same function.*

Then, considering the optimal hedge strategy, the author took the first-order Taylor expansion on the fair price $\widehat{C}$ (ignoring Remark (22.5.5)), getting

$$\widehat{C}_{t+1} = C_t + \frac{\partial C_t}{\partial X}\Delta X(t) + O(\Delta t)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Plugging the fair price expansion in the optimal strategy in Equation (22.5.26), we get

$$\alpha_t^* = \frac{Cov_t(C_t + \frac{\partial C_t}{\partial X}\Delta X(t), \Delta_r X(t))}{Var(\Delta_r X(t))}, t = T-1, ..., 0$$

Using the results in Appendix (24.1.3), since C_t and $\frac{\partial C_t}{\partial X}$ are known at time t, we get

$$\begin{aligned}\alpha_{t,BS}^* &= \frac{Cov_t(C_t, \Delta_r X(t))}{Var(\Delta_r X(t))} + \frac{Cov_t(\frac{\partial C_t}{\partial X}\Delta X(t), \Delta_r X(t))}{Var(\Delta_r X(t))} \\ &= 0 + \frac{\partial C_t}{\partial X}\frac{Cov_t(\Delta X(t), \Delta_r X(t))}{Var(\Delta_r X(t))}\end{aligned}$$

***Remark* 22.5.6**

$$Cov_t(\Delta X(t), \Delta_r X(t)) \neq Var_t(\Delta_r X(t))$$

since from Remark (22.5.1) $\Delta_r X(t) \neq \Delta X(t)$. *Thus,* $\alpha_{t,BS}^* \neq \frac{\partial C_t}{\partial X}$.

Nonetheless, taking the limit $\Delta t \to 0$, Halperin [2017] argued that the optimal strategy simplifies to

$$\alpha_t^{BS} = \lim_{\Delta t \to 0} \alpha_t^* = \frac{\partial C_t}{\partial X(t)}$$

which is the BS-delta.
Again, from Remark (22.5.1), this only occurs if $\mu = r$, and $E[\Delta_r X(t)|\mathcal{F}_t] = 0$. Thus, Equation (22.5.25) prevails and there is no extra term depending on the quantity of stock α. Nonetheless, focussing on Equation (22.5.24), the author computed the limit of the second term, getting

$$\lim_{\Delta t \to 0} \alpha(t) E_t[\Delta_r X(t)|\mathcal{F}_t] = \lim_{\Delta t \to 0} \alpha_{BS}(t) X(t)(\mu - r)dt = \lim_{\Delta t \to 0} X(t)\frac{\partial C_t}{\partial X(t)}(\mu - r)dt$$

Replacing the two terms above in Equation (22.5.24), the author observed that the drift term μ cancelled out and recovered the BS-PDE in Equation (22.6.30) under the risk-neutral measure $\mathbb{Q}$. This is a surprising result considering that the two original terms in Equation (22.5.24) were expressed under the historical measure $\mathbb{P}$. From Remarks (22.5.1) and (22.5.6), we clearly see that it is not the case.

22.6 Miscellaneous

22.6.1 Some facts on the BS-framework

22.6.1.1 Defining the dynamics of European option prices

We let $\{S_t; t \in T\}$ be a second-order stochastic process in the Hilbert space $L_2 = L_2(\Omega, \mathcal{F}, P)$ and assume that it represent the underlying asset. A self-financing portfolio is a strategy to buy or sell assets whose value is not modified by adding or removing cash. Its value at time t, and its variation in time between $(t, t+dt)$, satisfy

$$\begin{aligned}V_t &= n_t^0 S_t^0 + n_t^1 S_t^1 + ... + n_t^k S_t^k \\ dV_t &= n_t^0 dS_t^0 + n_t^1 dS_t^1 + ... + n_t^k dS_t^k\end{aligned}$$

where n_t^i is the quantity of stock S_t^i, for $i = 0, 1, ..., k$, in the portfolio.
We put ourself in the Black-Scholes world (see Black et al. [1973]) and model the dynamics of the future stock prices, under the historical measure $\mathbb{P}$, as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} dS_t &= S_t\big(\mu_S dt + \sigma_S d\widehat{W}_t\big) \\ \mu_S &= r + \sigma_S \lambda \end{aligned} \tag{22.6.27}$$

where r is the risk-free rate, σ_S is the volatility of the spot price, $\hat{W}_t$ is a Brownian motion, and λ is the market price of risk.
If we denote the quantity of cash that one can hold as $n_t^0 S_t^0$, and assume deterministic rates, its dynamics is given by

$$dS_t^0 = S_t^0 r dt$$

We now build a self-financing portfolio made of cash and stocks, and write the value invested in stocks as $\pi_t = n_t S_t$. The dynamics of the portfolio is

$$dV_t = n_t dS_t + n_t^0 dS_t^0$$

Replacing the respective dynamics in the above equation, the self-financing portfolio satisfies the equation

$$dV_t = rV_t dt + \pi_t \sigma_S (d\widehat{W}_t + \lambda dt)$$

We know that in a complete market, every terminal payoff can be replicated by a portfolio. This means that there exists a strategy leading to a final P/L of null value in all possible future configuration. We can conclude that the option price is equal to the value of the replicating portfolio at the initial time.
A direct application of the above theorem is that we can express the Black-Scholes price dynamics, under the historical measure $\mathbb{P}$, as

$$dC_{BS}(t, S_t) = rC_{BS}(t, S_t)dt + \pi_t \sigma_S (d\widehat{W}_t + \lambda dt)$$

We can also re-express that price dynamics under the risk-neutral measure by applying the change of measure (see Girsanov's theorem)

$$W_t = \widehat{W}_t + \int_0^t \lambda_s ds$$

where W_t is a $\mathbb{Q}$-Brownian motion. Therefore, the Black-Scholes price of a call option at all future time t is solution to the following stochastic equation with terminal condition

$$\begin{aligned} dC_{BS}(t, S_t) &= rC_{BS}(t, S_t)dt + \pi_t \sigma_S dW_t \\ C_{BS}(T, S_T) &= h(S_T) \end{aligned}$$

We are now going to decompose in a random function the Black-Scholes price $C_{BS}(t, S_t)$, taken as a function of class $C^{1,2}$ in t and S_t. Applying Ito's Lemma, the Black-Scholes price process satisfies the dynamics

$$dC_{BS}(t, S_t, K, T) = \frac{\partial C_{BS}(\cdot)}{\partial S_t} dS_t + \frac{\partial C_{BS}(\cdot)}{\partial t} dt + \frac{1}{2} \frac{\partial^2 C_{BS}(\cdot)}{\partial S_t^2} d < S_t > \tag{22.6.28}$$

Let's expand the dS_t term, we get

$$dC_{BS}(t, S_t, K, T) = \Big(\frac{1}{2}\sigma_S^2 S_t^2 \frac{\partial^2 C_{BS}(\cdot)}{\partial S_t^2} + rS_t \frac{\partial C_{BS}(\cdot)}{\partial S_t} + \frac{\partial C_{BS}(\cdot)}{\partial t}\Big) dt + \sigma_S S_t \frac{\partial C_{BS}(\cdot)}{\partial S_t} dW_t \tag{22.6.29}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The Black-Scholes formula [4] $C_{BS}(x;K,T-t;r,s)$ satisfies the PDE

$$\forall\, x>0 \quad \forall\, t<T \quad rC_{BS}(\cdot)=\frac{\partial C_{BS}(\cdot)}{\partial t}+\frac{1}{2}s^2x^2\frac{\partial^2 C_{BS}(\cdot)}{\partial x^2}+rx\frac{\partial C_{BS}(\cdot)}{\partial x} \tag{22.6.30}$$

with the terminal condition

$$\forall\, x>0 \quad C_{BS}(x;K,0;r,s)=(x-K)^+$$

Substituting Equation (22.6.30) into the Black-Scholes price process, we get

$$dBS(t,S_t,K,T)=rC_{BS}(\cdot)dt+\sigma_S S_t\frac{\partial C_{BS}(\cdot)}{\partial S_t}dW_t$$

such that the quantity invested in stock is given by

$$\pi_t=S_t\frac{\partial C_{BS}(\cdot)}{\partial S_t}$$

22.6.1.2 The Greeks and their interpretation

The Black-Scholes formula is a function

$$v(t,x,r,q;T,K;\sigma)$$

where the underlying asset $(S_t^x)_{t\geqslant 0}$ and the time t are state variables, the risk-free rate r, the repo rate q and the volatility σ are fixed parameters of the model, the maturity T and the strike K are fixed parameters of the contingent claim. The sensitivities of a call price with respect to these parameters play a fundamental role in the risk management of derivative products. For instance, a call option with maturity T and strike K can be hedged with the quantity

$$\Delta(t,S_t)=\partial_x C_{BS}(t,S_t,K,T)=e^{-q(T-t)}N(d_1(T-t,F(t,T),K))$$

of the risky asset. For simplicity we let $t=0$ with initial value $S_0=x$ and we define $n(x)=\partial_x N(x)$. In the following we briefly present a few Greeks:

$$\begin{aligned}
\partial_{xx}C &= \Gamma=\frac{e^{-q(T-t)}}{x\sigma\sqrt{T-t}}n(d_1)>0\\
\partial_\sigma C &= Vega=xe^{-q(T-t)}\sqrt{T-t}n(d_1)\\
\partial_K C &= -e^{-r(T-t)}N(d_2)\\
\partial_r C &= \rho=(T-t)Ke^{-r(T-t)}N(d_2)>0\\
\partial_t C &= \Theta=-\frac{x\sigma}{2\sqrt{T-t}}e^{-q(T-t)}n(d_1)+qxe^{-q(T-t)}N(d_1)-rKe^{-r(T-t)}N(d_2)
\end{aligned}$$

and the Black-Scholes partial differential equation (PDE) can be written as

$$\frac{1}{2}\sigma^2x^2\Gamma+(r-q)x\Delta-rC+\Theta=0 \tag{22.6.31}$$

Since we can express the Gamma in terms of the Vega, we can rewrite the PDE as

$$\frac{\sigma}{2T}Vega+(r-q)x\Delta-rC+\Theta=0$$

[4] $C_{BS}(x;K,T-t;r,s)=x\mathcal{N}\left(\frac{\ln(\frac{x}{K})+(r+\frac{1}{2}s^2)(T-t)}{s\sqrt{T-t}}\right)-Ke^{-r(T-t)}\mathcal{N}\left(\frac{\ln(\frac{x}{K})+(r-\frac{1}{2}s^2)(T-t)}{s\sqrt{T-t}}\right)$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

While $\Delta(t, S_t)$ is a quantity of the risky asset to buy or sell when building the replicating portfolio, the other Greeks are not easily interpreted as financial quantities and must therefore be modified for risk management purposes. Assuming constant volatility, the dynamics of the stock price in discrete time is given by

$$\Delta S_t = \mu S_t \Delta t + \sigma S_t y \sqrt{\Delta t}$$

where ΔS_t is the variation of the stock price in the time interval Δt, and $y \sim N(0,1)$. As a result, $\frac{\Delta S_t}{S_t}$ follows a normal distribution with mean $\mu \Delta t$ and standard deviation $\sigma\sqrt{\Delta t}$. That is, $\frac{\Delta S_t}{S_t} \sim N(\mu\Delta t, \sigma\sqrt{\Delta t})$. We let $S^+ = S + \alpha$ where $\alpha = \epsilon S$ with $\epsilon << 1$, and we define a change in spot as $\Delta S = S^+ - S = \alpha$. In general, the market assumes $\epsilon = 1\%$. Using finite difference, we can approximate the Delta with a one-sided finite difference as

$$\Delta \approx \frac{C(S^+) - C(S)}{\alpha}$$

In order to include some convexity in the delta, by considering the difference in value between an option after an upward bump and one after a downward bump, we can use a two-sided finite difference as

$$\Delta \approx \frac{C(S^+) - C(S^-)}{2\alpha}$$

In the case of the one-sided delta we can deduce that

$$C(S^+) \approx \alpha\Delta + C(S)$$

We see that we can obtain a modified Delta as $\widehat{\Delta} = \alpha\Delta$ such that

$$C(S^+) \approx \widehat{\Delta} + C(S)$$

where we get the new price simply by adding $\widehat{\Delta}$ to the old price. Similarly, we can rewrite the Gamma in terms of the Delta as

$$\Gamma \approx \frac{\Delta(S^+) - \Delta(S)}{\alpha}$$

and deduce that

$$\Delta(S^+) \approx \alpha\Gamma + \Delta(S)$$

We see that we can obtain a modified Gamma as $\widehat{\Gamma} = \alpha\Gamma$ such that

$$\Delta(S^+) \approx \widehat{\Gamma} + \Delta(S)$$

where we get the new Delta simply by adding $\widehat{\Gamma}$ to the old Delta. In other words, if the spot price jumps from S to S^+, the trader must immediately modify his position in the risky asset by buying the quantity $\widehat{\Gamma}$ (or selling that quantity if $\widehat{\Gamma} < 0$) of the risky asset. Hence, the modified Gamma corresponds to a change in the hedging strategy induced by a sudden jump in the underlying. We now let $\sigma^+ = \sigma + \alpha$ where $\alpha = \epsilon\sigma$ with $\epsilon << 1$, and we define a change in volatility as $\Delta\sigma = \sigma^+ - \sigma = \alpha$. Using one-sided finite difference, we can approximate the Vega as

$$Vega \approx \frac{C(\sigma^+) - C(\sigma)}{\alpha}$$

Since volatility is already expressed in percentage, market practice is to directly set $\alpha = 1\%$ and obtain the modified Vega as $\widehat{V}ega = \alpha Vega$ such that

$$C(\sigma^+) \approx \widehat{V}ega + C(\sigma)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where we get the new price simply by adding $\widehat{V}ega$ to the old price. While theoretically the vega does not exist in the BS-formula, it provides an exposition of the call price to volatility risk. This is related to the IV surface and the fact that traders acknowledge a stochastic variation of the volatility. At last, the Theta above is calculated with yearly parameters and represents time decay over a yearly period. To express time decay over a one day period, we simply multiply the Theta with a fraction of time, getting

$$\widehat{\Theta} = \Theta \times \Delta t$$

where $\Delta t = \frac{1}{365}$.

22.6.2 Some expansions

22.6.2.1 Taylor series expansion

22.6.2.1.1 Ordinary calculus The second order Taylor series for $f(t,x)$ around the point (a,b) is

$$f(t,x) = f(a,b) + f_t(a,b)(t-a) + f_x(a,b)(x-b) + \frac{1}{2}\Big[f_{tt}(a,b)(t-a)^2 + 2f_{tx}(a,b)(t-a)(x-b) + f_{xx}(x-b)^2\Big] + \cdots \tag{22.6.32}$$

In order to take the derivative with respect to t or x, we let let a and b approach t and x, respectively. In the limit, all the second order and higher terms tend to zero much faster than the first order terms, and are dropped. As a result, in the limit, we can rewrite the above equation in differential form as

$$df(t,x) = \frac{\partial f(t,x)}{\partial t}dt + \frac{\partial f(t,x)}{\partial x}dx$$

However, with stochastic processes, the dx term actually contains something on the order of $\sqrt{dt}$. Thus, the $(dx)^2$ term contains something of the order dt, and so on. Hence, when taking the limit, the equation contains a term which does not tend to zero any faster than dx, and as such, is not dropped.

In multi-dimension, we let the function be denoted by $V(t,X)$, where t is the time and X is the explanatory factor, such that $V : \mathbb{R}^{n+1} \to \mathbb{R}$, and assume that it is infinitely differentiable at a real point in the neighbourhood of (t,X). Further, we let $X^* = X + \Delta X$ and apply a (second-order) Taylor series expansion on V with respect to time and the vector of factor, getting the price variation

$$\Delta V_t = \frac{\partial V}{\partial t}\Delta t + \nabla V(t,X)\cdot(X^* - X) + \frac{1}{2}(X^* - X)^\top HV(t,X)(X^* - X) + R_2(X,X^*) \tag{22.6.33}$$

where $(X^* - X)$ is an n-dimensional column vector, $\nabla V = DV = \left(\frac{\partial V}{\partial x_1}(\cdot), ..., \frac{\partial V}{\partial x_n}(\cdot)\right)$ is the gradient vector, and H is the Hessian matrix at the point X. Note, the error term $R_2(X,X^*)$ is going to zero faster than a constant times $\|X^* - X\|^3$ as $X^* \to X$.

22.6.2.1.2 Informal derivation Assume that X_t is an Ito-diffusion process satisfying the SDE

$$dX_t = \mu_t dt + \sigma_t dW_t$$

where W_t is a Wiener process. If $f(t,x)$ is a twice-differentiable scalar function, its Taylor series expansion is

$$df = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial x}dx + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}dx^2 + ...$$

Substituting X_t for x, we get

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$df = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial x}\big(\mu_t dt + \sigma_t dW_t\big) + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}\big(\mu_t^2(dt)^2 + 2\mu_t\sigma_t dt dW_t + \sigma_t^2(dW_t)^2\big) + ...$$

In the limit $dt \to 0$, the terms $(dt)^2$ and $dtdW_t$ tend to zero faster than $(dW_t)^2$, which is $O(dt)$. Thus, setting the terms $(dt)^2$ and $dtdW_t$ to zero, substituting dt for $(dW_t)^2$, and collecting the terms dt and dW_t, we get

$$df = (\frac{\partial f}{\partial t}dt + \mu_t\frac{\partial f}{\partial x} + \frac{1}{2}\sigma_t^2\frac{\partial^2 f}{\partial x^2})dt + \sigma_t\frac{\partial f}{\partial x}dW_t \tag{22.6.34}$$

as required.

22.6.2.2 Ito's formula

We recall Ito's formula for semi-martingales in the multi-dimension case.

***Proposition* 12** *Let $X_t = (X_1(t), ..., X_d(t))$ to be a multi-dimensional semi-martingale. Then for any $C^{1,2}$ function $f : [0,T] \times \mathbb{R}^d \to \mathbb{R}$*

$$\begin{aligned} f(t,X_t) - f(0,X_0) &= \int_0^t \sum_{i=1}^d \frac{\partial f}{\partial x_i}(s,X_{s^-})dX_i(s) + \int_0^t \sum_{i=1}^d \frac{\partial f}{\partial s}(s,X_s)ds \\ &+ \frac{1}{2}\int_0^t \sum_{i,j=1}^d \frac{\partial^2 f}{\partial x_i \partial x_j}(s,X_s)d[X_i,X_j]_s^C \\ &+ \sum_{\substack{s\in[0,t] \\ \Delta X_s \neq 0}} \Big[f(s,X_{s^-}+\Delta X_s) - f(s,X_{s^-}) - \sum_{i=1}^d \Delta X_i(s)\frac{\partial f}{\partial x_i}(s,X_{s^-})\Big] \end{aligned}$$

22.6.3 The PAA methodology

In order to reconcile the pseudo continuous hedging performed by traders and the necessity by market risk to assess model risk, the latter assumes the option price to be a function of some explanatory factors, and apply a Taylor expansion over a small period of time. The aim being to explain the variation of market prices over a small period of time with some explanatory factors. This method for assessing the ability of a pricing model to reproduce past market observable is called Profit Attribution Analysis (PAA). The idea being that in the limit, the dynamics implied by the PAA should not be far from that of option prices computed with a pricing model using stochastic calculus and the theory of no-arbitrage.

22.6.3.1 Definition

We let T be the domain of definition taking the discrete values $t_1, ..., t_n$. At every fixed date t_i, $i = 1, ..., n$, we compute the option prices. We let the claim be denoted by $V(t,X)$, where t is the time and X is the explanatory factor, such that $V : \mathbb{R}^{d+1} \to \mathbb{R}$, and assume that it is infinitely differentiable at a real point in the neighbourhood of (t,X). Further, we let $X^* = X + \Delta X$ and apply a (second-order) Taylor series expansion on V with respect to time and the vector of factor (see Appendix (22.6.2.1)), getting the price variation

$$\Delta V_t = \frac{\partial V}{\partial t}\Delta t + \sum_{i=1}^d \frac{\partial V}{\partial x_i}(t,X)(x_i^* - x_i) + \frac{1}{2}\sum_{i,j=1}^d \frac{\partial^2 V}{\partial x_i \partial x_j}(t,X)(x_i^* - x_i)(x_j^* - x_j) + R_2(X,X^*) \tag{22.6.35}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where the error term $R_2(X, X^*)$ is going to zero faster than a constant times $\|X^* - X\|^3$ as $X^* \to X$.
We define the risked-base PAA as

$$PAA(t) = \sum_{i=1}^{d} \frac{\partial V}{\partial x_i}(t, X)(x_i^* - x_i) + \frac{1}{2} \sum_{i,j=1}^{d} \frac{\partial^2 V}{\partial x_i \partial x_j}(t, X)(x_i^* - x_i)(x_j^* - x_j)$$

The equivalent of this change in price using the theory of option pricing can be obtained by applying Ito's formula for semi-martingales in the multi-dimension case on option prices (see Proposition (12)). For comparison, we reproduce the formula in the time interval $[t, t^*]$ where $t^* = t + \Delta t$, and for the explanatory factor $X_t = (X_1(t), ..., X_d(t))$.

$$\begin{aligned} V(t^*, X_{t^*}) - V(t, X_t) &= \int_t^{t^*} \sum_{i=1}^{d} \frac{\partial V}{\partial x_i}(s, X_{s^-}) dX_i(s) + \int_t^{t^*} \sum_{i=1}^{d} \frac{\partial V}{\partial s}(s, X_s) ds \\ &+ \frac{1}{2} \int_t^{t^*} \sum_{i,j=1}^{d} \frac{\partial^2 V}{\partial x_i \partial x_j}(s, X_s) d[X_i, X_j]_s^C \\ &+ \sum_{\substack{s \in [t, t^*] \\ \Delta X_s \neq 0}} \left[V(s, X_{s^-} + \Delta X_s) - V(s, X_{s^-}) - \sum_{i=1}^{d} \Delta X_i(s) \frac{\partial V}{\partial x_i}(s, X_{s^-}) \right] \end{aligned}$$

22.6.3.2 PAA limitations

Note, the portfolio variation ΔV_t in the PAA does not correspond to the delta-hedged profit and loss via the theory of no-arbitrage (see details in Section (22.6.1.1)).
Further, there are some limitations to computing the sensitivities of prices (see Tankov et al. [2010]), some of which being:

1. they only relate changes in the values of derivative products to the changes of basic risk factors.
2. they can not be aggregated across different underlyings and across different types of sensitivities (delta plus gamma). They are local measures and do not provide global information about the entire portfolio.
3. they are only meaningful for small changes of risk factors, and do not provide accurate information about the change of portfolio to larger moves (jumps).
4. the sensitivity to a given risk factor is associated to a very specific scenario of market dynamics applying only to the change in that risk factor while others are kept constant.

If we compare the gradient in the two methods, we see that

- the sensitivity $\frac{\partial V}{\partial x_i}$ is computed at time t in the PAA and held constant in the interval Δt, while in theory, it should be recalculated continuously with a pricing model.
- the sensitivity $\frac{\partial V}{\partial x_i}$ is computed at all time t in the PAA, even if the factor is discontinuous, leading to potentially large errors. This is not the case in a pricing model, when assuming a jump process in the dynamics of the factors.

If we compare the Hessian matrix at the point X in the two methods, we see that

- the sensitivity $\frac{\partial^2 V}{\partial x_i \partial x_j}$ is computed at time t in the PAA and held constant in the interval Δt, while in theory, it should be recalculated continuously with a pricing model.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- the sensitivity $\frac{\partial^2 V}{\partial x_i \partial x_j}$ is computed at all time t in the PAA, even if the factor is discontinuous, leading to potentially very large errors. This is not the case in a pricing model, when assuming a jump process in the dynamics of the factors. In that case, the sensitivity is only computed on the increasing continuous process, not on the increasing pure jump process.

22.6.3.3 Example

In order to illustrate the PAA, we assume that an option price V is composed of three explanatory variables, the time t, the stock price S, and the volatility σ. In that setting, we let $V = V(t, S, \sigma)$ be the option price, where σ represents a stochastic volatility process, and we apply a Taylor expansion on ΔV with respect to the time, the stock and the volatility, getting the variation

$$\Delta V_t = \frac{\partial V}{\partial t}\Delta t + \frac{\partial V}{\partial S}\Delta S_t + \frac{\partial V}{\partial \sigma}\Delta\sigma + \frac{1}{2}\Big[\frac{\partial^2 V}{\partial S^2}(\Delta S_t)^2 + \frac{\partial^2 V}{\partial \sigma^2}(\Delta\sigma)^2 + 2\frac{\partial^2 V}{\partial S \partial \sigma}(\Delta S_t)(\Delta\sigma)\Big] + \epsilon(t) \tag{22.6.36}$$

where $\Delta V_t = V(t + \Delta t, S + \Delta S, \sigma + \Delta\sigma) - V(t, S, \sigma)$. It is a quadratic function of ΔS_t where the linear terms behaves like the stock price and the quadratic term increases no matter how S_t moves. We can rewrite the portfolio variation as

$$\Delta V_t = \widehat{\Theta} + \widehat{\Delta} + \widehat{V}ega + \frac{1}{2}\widehat{\Gamma}\Delta S_t + \frac{1}{2}Volga(\Delta\sigma)^2 + Vanna(\Delta S_t)(\Delta\sigma) + \epsilon(t) \tag{22.6.37}$$

where $Volga = \frac{\partial^2 V}{\partial \sigma^2}$ and $Vanna = \frac{\partial^2 V}{\partial S \partial \sigma}$. Note, if $\epsilon = \frac{\Delta S}{S}$ we get $\widehat{\Delta} = (\frac{\Delta S}{S})S\Delta(t, S_t)$ where $S\Delta(t, S_t)$ is the equivalent stock position (ESP), or dollar delta. We see that the knowledge of the modified Greeks allows the traders to estimate the variation of the claim over a small period of time in the case where the stock price follows a diffusion process with stochastic volatility. More precisely, this portfolio variation assumes a one day period, with a spot price change of 1% and a volatility change of 1%, which is not very realistic.

22.6.3.4 Market practices

In the simple case where we let the stock price be driven by the BS-model, the dynamics of the stock price is given in Equation (22.6.27), and the BS-price process is given in Equation (22.6.28).
Following the PAA analysis, we let the claim be denoted $V(S, t)$, and apply a Taylor series expansion on ΔV with respect to the time, and the stock price, getting the variation

$$\Delta V_t = \frac{\partial V}{\partial t}\Delta t + \frac{\partial V}{\partial S}\Delta S_t + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}(\Delta S_t)^2 + ...$$

In the limit, when Δt tends towards an infinitesimal quantity, this expansion corresponds to the BS-price process in Equation (22.6.28).
Note, so far the dynamics of the option price occurs under the historical measure $\mathbb{P}$.
Note, the market introduces an additional sensitivity to the equity volatility σ_S in the PAA analysis, even though the variation of the volatility is equal to zero. Thus, the Taylor series expansion on ΔV with respect to the time, and the stock price, becomes

$$\Delta V_t = \frac{\partial V}{\partial t}\Delta t + \frac{\partial V}{\partial S}\Delta S_t + \underbrace{\frac{\partial V}{\partial \sigma}\Delta\sigma}_{\text{volatility}} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}(\Delta S_t)^2 + ...$$

It is a first step to acknowledging that traders using the BS-formula keep changing the BS-volatility for different strikes and maturities (K, T). Thus, we can see it as a first order approximation to the BS-price process given when accounting for implied volatility $I(t, T)$, where

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\frac{1}{2}\frac{\partial^2 BS()}{\partial\sigma^2}d < I(t,T) > + \frac{\partial^2 BS()}{\partial S_t \partial\sigma}d < S_t, I(t,T) >= 0$$

The problem becomes that of computing the variation $dI(t,T)$ over a small time increment.
If we now let the stock price be driven by a stochastic volatility model, the option variation in Equation (22.6.36) could be compared to the BS-price process when accounting for implied volatility if we assumed

$$\frac{\partial V}{\partial\sigma}\Delta\sigma \approx \frac{\partial C_{BS}()}{\partial\Sigma}d\Sigma(t;K,T-t)$$

However, we can not find an implied volatility surface such that the no-arbitrage condition is satisfied.

22.6.4 Point on Kolmogorov-Compatibility

Dealing with deterministic smile surface evolution always rises the question of whether the smile surface allows for arbitrage opportunities (see Section (22.1.1.5)). It is natural to impose that future smile surface are compatible with today's prices of calls and puts. The problem is that defining a deterministic smile surface $\sigma(t,T,K,S_t)$ means defining a future deterministic density ϕ for stock process, and then a natural condition would be to impose that the future pdf is actually a conditional density. This is what we will formalise as the Kolmogorov-Compatibility. First of all, let's remind the relation between prices and pdf. Without doing any assumptions apart from that of no arbitrage opportunity, we have

$$\begin{aligned}
C(t,K,t_1-t_0) &= e^{-r(t_1-t_0)}\left((S_{t_1}-K)^+|\mathcal{F}_{t_1}\right) = e^{-r(t_1-t_0)}\int_K^{+\infty}(x_1-K)\phi(x_1,t_1,S_{t_0},t_0)d\lambda(x_1)\\
\frac{\partial C}{\partial K}(t,K,t_1-t_0) &= -e^{-r(t_1-t_0)}\int_K^{+\infty}\phi(x_1,t_1,S_{t_0},t_0)d\lambda(x_1)\\
\frac{\partial^2 C}{\partial K^2}(t,K,t_1-t_0) &= e^{-r(t_1-t_0)}\phi(K,t_1,S_{t_0},t_0)
\end{aligned}$$

By definition $C(t,K,t_1-t_0) = C_{BS}(S_{t_0};K,t_1-t_0;r,\Sigma(S_{t_0};K,t_1-t_0))$, such that applying the chain rule we have

$$\frac{\partial^2 C}{\partial K^2} = \frac{\partial^2 C_{BS}}{\partial K^2} + \frac{\partial^2 C_{BS}}{\partial K\partial\Sigma}\frac{\partial\Sigma}{\partial K} + \left[\frac{\partial^2 C_{BS}}{\partial K\partial\Sigma} + \frac{\partial^2 C_{BS}}{\partial K^2}\frac{\partial\Sigma}{\partial K}\right]\frac{\partial\Sigma}{\partial K} + \frac{\partial C_{BS}}{\partial\Sigma}\frac{\partial^2\Sigma}{\partial K^2}$$

For the sake of clarity we denote by Θ the right-hand-side operator.

Definition 22.6.1 (Kolmogorov-Compatibility) *Any future deterministic conditional density, or smile surface, such that*

$$\begin{aligned}
&\Theta(S_0;K,T-t_0;r,\sigma(t_0,T,K,S_{t_0})) =\\
&\int\Theta(S_0;K,t_1-t_0;r,\sigma(t_0,t_1,K,S_{t_0}))\Theta(S_{t_1};K,T-t_1;r,\sigma(t_1,T,K,S_{t_1}))d\lambda(S_{t_1})
\end{aligned}$$

is satisfied, defines a Kolmogorov-compatible density.

Property 22.6.1 *Given a current admissible*[5] *smile surface, if the future smile surface is Kolmogorov-compatible no model-independent strategy can generate arbitrage profits.*

[5] *i.e.* such that the associated call prices sarisfy $\frac{\partial Call(t,K,T-t)}{\partial K} < 0$, $\frac{\partial^2 Call(t,K,T-t)}{\partial K^2} > 0$, $\frac{\partial Call(t,K,T-t)}{\partial T} > 0$, $\frac{\partial Put(t,K,T-t)}{\partial K} > 0$, $Call(t,K,T-t)|_{K=0} = S_t$ and $lim_{K\to\infty}Call(t,K,T-t) = 0$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

There is in general an infinity of solutions for the forward density

$$\Theta(S_{t_1}; K, T - t_1; r, \sigma(t_1, T, K, S_{t_1}))$$

such that Definition (22.6.1) is satisfied. Therefore, even if we require the smile surface to be deterministic, there still exists an infinity of future smile surfaces compatible with today's prices of calls and puts. Additional information is needed to determine these conditional distributions. We need to know all forward smiles, that is, the future prices of all vanilla options as seen from all possible states of the world.

22.7 The parametric MixVol model

Bloch [2010] [2012] considered a parametric interpolation and extrapolation of the implied volatility (IV) surface by decomposing the market option prices into a weighted sum of strike shifted Black-Scholes counterparts. A term structure of volatility was introduced to infer plausible deformation of the IV surface both in space and in time. A Differential Evolution algorithm was used to solve a non-linear optimisation problem under constraints. We are now going to describe that model.

22.7.1 Description of the model

To obtain a pronounced skew we consider a sum of shifted log-normal distributions, that is, using the Black-Scholes formula with shifted strike (modified by the parameters $\mu_i(t)$) as an interpolation function. In our parametric model, the market option price $C_M(K,t)$, for a strike K and maturity t, is estimated at time $t_0 = 0$ by the weighted sum

$$C_M(t_0, S_0, P_t, R_t, D_t; K, t) = \sum_{i=1}^{n} a_i(t) Call_{BS}(t_0, S_0, R_t, P_t, \overline{K}(K,t), t, \Sigma_i(t)) \tag{22.7.38}$$

where $a_i(t)$ for $i = 1, .., n$ are the weights, $\overline{K}(K,t) = K^{'}(K,t)(1 + \mu_i(t))$ with $K^{'}(K,t) = K + D_t$. In that setting $R_t = Re(0,t)$ is the repo factor in the range $[0,t]$, $P_t = P(0,t) = e^{-rt}$ is the zero-coupon bond price, $C_t = C(0,t) = \frac{R_t}{P_t}$ is the cost of carry and $D_t = D(0,t)$ is the compounded sum of discrete dividends between $[0,t]$. We let the time function $t \to \Sigma_i(t)$ be regular enough, and choose to model directly the square-root of the average variance defined as

$$\Sigma_i^2(t) = \frac{1}{t} \int_0^t \sigma^2(s) ds$$

where $\sigma(t)$ is the instantaneous volatility of the underlying process. To guarantee the positivity of the local volatility, the average variance must verify

$$\Sigma_i^2(t) + 2\Sigma_i(t) t \partial_t \Sigma_i(t) \geqslant 0 \, , \forall i$$

and since $\Sigma_i(t) > 0$, the constraint simplifies to

$$\Sigma_i(t) \geqslant -2t\partial_t \Sigma_i(t) \, , \forall i \tag{22.7.39}$$

Further, the no-arbitrage theory imposes time and space constraints on market prices. Introducing the time dependent parameters $a_i(t)$ and $\mu_i(t)$, the simplest way of ensuring these constraints is to take the same time dependency for each μ, that is, $\mu_i(t) = \mu_i f(t, \beta_i)$. As the pdf of the equity price should tend toward a single Dirac when $t \to 0$, to get control on $\mu_i(t)$, we choose to let the function $f(t,x)$ tend to 1 when $t \to +\infty$, getting

$$f(t,x) = 1 - \frac{2}{1 + (1 + \frac{t}{x})^2}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

with $f^{'}(t,x) = [1 - f(t,x)]^2 \frac{1}{x}(1 + \frac{t}{x})$. Moreover, to keep manageable the no-free lunch constraints, we make the weight $a_i(t)$ proportional to $\frac{a_i^0}{f(t,\beta_i)}$ for some constant $a_i^0 > 0$, getting the representation

$$\mu_i(t) = \mu_i^0 f(t,\beta_i) \text{ and } a_i(t) = \frac{a_i^0}{f(t,\beta_i) \times norm}$$

where $norm = \sum_{i=1}^n \frac{a_i^0}{f(t,\beta_i)}$. As a result, with seperable functions of time, we can prove that the no-free lunch constraints simplify.

Theorem 22.7.1 *No-Arbitrage Constraints in the MixVol Model*
We let the price of a European call option be given by Equation (22.7.38) and assume seperable functions of time $a_i(t)$, $\mu_i(t)$. Then the resulting implied volatility surface is free from static arbitrage (according to Definition (22.1.5)) provided the following parameter restrictions hold:

$$\begin{aligned} a_i^0 &\geqslant 0 \\ \sum_{i=1}^n a_i(t) &= 1 \\ \sum_{i=1}^n a_i^0 \mu_i^0 &= 0 \\ \mu_i^0 &\geqslant -1 \end{aligned} \tag{22.7.40}$$

To make sure that the IV surface flattens for infinitely large expiries, we impose the limit behaviour

$$\lim_{t\to\infty} \Sigma_i(t) = d_i \tag{22.7.41}$$

where $d_i > 0$. For example, Bloch [2010] discussed the volatility function

$$\Sigma_i(t) = \big(a_i + b_i t\big)e^{-c_i t} + d_i$$

where one must impose $b_i \geqslant 0$ for $\Sigma_i(.)$ to remain positive, generating only upward humps.
To get a general volatility function capable of generating both an upward hump or a downward one, Bloch [2012] proposed the volatility function

$$\Sigma_i(t) = \big(a_i + b_i \ln(1 + e_i t)\big)e^{-c_i t} + d_i$$

where $c_i > 0$, $d_i > 0$ and $a_i \in \mathbb{R}$, $b_i \in \mathbb{R}$ and $e_i \in]-\frac{1}{t}, \infty[$. The derivative of the function with respect to time t is

$$\Sigma_i^{'}(t) = \big(-a_i c_i + b_i \frac{e_i}{1 + e_i t} - b_i c_i \ln(1 + e_i t)\big)e^{-c_i t}$$

and the no-arbitrage constraint must satisfy

$$e^{-c_i t}\big[(1 - 2tc_i)\big(a_i + b_i \ln(1 + e_i t)\big) + 2tb_i \frac{e_i}{1 + e_i t}\big] + d_i \geqslant 0$$

Since $a_i \in \mathbb{R}$, at time $t = 0$ the left hand side of the inequality can becomes negative. Consequently, we must impose the constraint

$$a_i + d_i > 0$$

to get the constraint satisfied at $t = 0$. Further, when $t > \frac{1}{c_i}$ then for c_i sufficiently large the constant d_i will dominates $\big(a_i + b_i \ln(1 + e_i t)\big)$ ensuring positivity of the left hand side. Hence, at time $t = \frac{1}{c_i}$ we must impose

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$d_i \geqslant \big(a_i - 2\frac{b_i}{c_i}\frac{e_i}{1+\frac{e_i}{c_i}} + b_i \ln\big(1+\frac{e_i}{c_i}\big)\big)e^{-1}$$

22.7.2 Computing the Greeks

Assuming that the dynamics of the spot price with discrete dividends follow the Spot model, our parametric model in Equation (22.7.38) corresponds to a weighted sum of Black-Scholes formulas in the Z-space. In that setting, the derivative of a call option price with respect to the spot is

$$\frac{\partial}{\partial S_0}C_M(t_0, S_0, P_t, R_t, D_t; K, t) = \frac{1}{norm}\sum_{i=1}^{n}\overline{a}_i(t)\frac{\partial}{\partial S_0}Call_{BS}(t_0, S_0, R_t, 1, \tilde{K}(K,t), t, \Sigma_i(t))$$

where $\tilde{K}(K,t) = P(t_0,t)\overline{K}(K,t)$ and $\overline{K}(K,t) = K^{'}(K,t)(1+\mu_i(t))$ and $\overline{a}_i(t) = \frac{a_i^0}{f(t,\beta_i)}$. Differentiating one more time the call price with respect to the spot, we get

$$\frac{\partial^2}{\partial S_0^2}C_M(t_0, S_0, P_t, R_t, D_t; K, t) = \frac{1}{norm}\sum_{i=1}^{n}\overline{a}_i(t)\frac{\partial^2}{\partial S_0^2}Call_{BS}(t_0, S_0, R_t, 1, \tilde{K}(K,t), t, \Sigma_i(t))$$

Differentiating the call price with respect to the strike, we get

$$\frac{\partial}{\partial K}C_M(t_0, S_0, P_t, R_t, D_t; K, t) = \frac{1}{norm}\sum_{i=1}^{n}\overline{a}_i(t)\frac{\partial}{\partial \tilde{K}}Call_{BS}(t_0, S_0, R_t, 1, \tilde{K}(K,t), t, \Sigma_i(t))\frac{\partial \tilde{K}}{\partial K} \tag{22.7.42}$$

where $\frac{\partial \tilde{K}}{\partial K} = P(t_0,t)(1+\mu_i(t))$.

22.7.3 Scenarios analysis

We are going to present a few stress scenarios defined on the IV surface to visualise and quantify the risk inherent to option portfolios. We choose to consider a range of plausible scenarios for the dynamics of the implied volatility surface. To do so we will modify the model parameters such that the the no-arbitrage constraints are satisfied. In order to obtain an upward or downward movement of the skew and curvature on the IVS, we are going to bump each function's curve $f_i(t)$ inside the shift function $\mu_i(t)$ with a constant. As a result, one of the shifted parametric price becomes

$$C_M(t_0, S_0, P_t, R_t, D_t; K, t; \epsilon) = \frac{1}{norm}\sum_{i=1}^{n}\overline{a}_i(t)Call_{BS}(t_0, S_0, R_t, P_t, K^{'}(K,t)(1+\mu_i^0(f_i(t)+\epsilon)), t, \Sigma_i(t))$$

Rather than modifying the function $f_i(t)$, we can directly modify the parameter β_i in the previous parametric price $C_M(t_0, S_0; K, t)$, getting $f(t, \beta_i(1+\epsilon))$ with $\epsilon << 1$. In addition, multiplying each parameter μ_i^0 with the same constant $(1+\epsilon)$, rather than adding a constant, will preserve the no-arbitrage constraints. Hence, under that scenario, the shifted option price becomes

$$C_M(t_0, S_0, P_t, R_t, D_t; K, t; \epsilon) = \frac{1}{norm}\sum_{i=1}^{n}\overline{a}_i(t)Call_{BS}(t_0, S_0, R_t, P_t, K^{'}(K,t)(1+(1+\epsilon)\mu_i^0 f_i(t)), t, \Sigma_i(t))$$

where $\epsilon << 1$. Alternatively, we can multiply each parameter μ_i^0 with the constant $(1+\epsilon_i)$ if we force one degree of liberty as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\epsilon_1 = -\frac{1}{a_1^0 \mu_1^0} \sum_{i=2}^{n} a_i^0 \mu_i^0 (1 + \epsilon_i)$$

One can also bump each μ_i^0 with ϵ_2 for $i = 1, .., n$, but in order to satisfy the no-arbitrage constraint we also need to bump each weight a_i^0 with ϵ_1 such that

$$\epsilon_1 = -\frac{\epsilon_2}{\sum_{i=1}^{n} \mu_i^0 + n\epsilon_2}$$

As the representation of the IVS is a highly non-linear function of the term structures $\Sigma_i(t)$ as well as the shift parameters $\mu_i(t)$, adding a positive constant ϵ to each term structure $\Sigma_i(t)$ will produce a parallel shift in the MixVol term structure $\widehat{\sigma} = \sum_{i=1}^{n} a_i(t)\Sigma_i(t)$ but not in the IVS itself. That is, for $t_1 > t_0$ the movement of the new implied volatility surface $\Sigma(t_1; K^{'}, t)$ produced by that scenario depends on the shape of the IVS at inception $\Sigma(t_0; K^{'}, t)$ which is in accordance with the result found by Rogers et al. [2010]. The smaller the skew, the larger the impact of $\widehat{\sigma}$ on the IV surface. As a result, we can shift each BS function $\Sigma_i(t)$ with a positive constant to generate a parallel shift type of movements of the IVS around the money, getting the shifted parametric price

$$C_M(t_0, S_0, P_t, R_t, D_t; K, t; \epsilon) = \frac{1}{norm} \sum_{i=1}^{n} \overline{a}_i(t) Call_{BS}(t_0, S_0, R_t, P_t, \overline{K}(K, t), t, \Sigma_i(t) + \epsilon)$$

where $\epsilon = 1\%$ to be consistent with the Black-Scholes vega. We can also define another type of volatility scenario dependent on the maturity with

$$(1 + \epsilon)\Sigma_i^2(t) = (1 + \epsilon)\frac{1}{t}\int_0^t \sigma^2(s)ds$$

where $\epsilon << 1$, so that the shifted option price becomes

$$C_M(t_0, S_0, P_t, R_t, D_t; K, t; \epsilon) = \frac{1}{norm} \sum_{i=1}^{n} \overline{a}_i(t) Call_{BS}(t_0, S_0, R_t, P_t, \overline{K}(K, t), t, \Sigma_i(t)(1 + \epsilon))$$

Combining all these bumps together, we obtain a no-arbitrage deformation of the IV surface generated by the shifted option price

$$\begin{aligned} & C_M(t_0, S_0, P_t, R_t, D_t; K, t; \epsilon_\mu, \epsilon_\beta, \epsilon_\sigma) \\ = \quad & \frac{1}{norm} \sum_{i=1}^{n} \overline{a}_i(t) Call_{BS}(t_0, S_0, R_t, P_t, K^{'}(K, t)(1 + (1 + \epsilon_\mu)\mu_i^0 f(t, \beta_i(1 + \epsilon_\beta))), t, \Sigma_i(t) + \epsilon_\sigma) \end{aligned}$$

For risk management purposes, we can therefore compute the modified vega as

$$\widehat{V}ega = C_M(t_0, S_0, P_t, R_t, D_t; K, t; \epsilon_\mu, \epsilon_\beta, \epsilon_\sigma) - C_M(t_0, S_0, P_t, R_t, D_t; K, t)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Part VII

Annexes

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 23

Some mathematical facts

23.1 Notation

We let $\mathbb{R}^n$ be the n-dimensional Euclidean space, $\mathbb{R}_+$ be the non-negative real numbers, $\mathbb{Q}$ the rational numbers, $\mathbb{Z}$ the integers, $\mathbb{Z}_+ = \mathbb{N}$ the natural numbers, $\mathbb{C}$ the complex plane, and $R^{n\times m}$ the $n \times m$ matrices with real entries.
We let $C(U, V)$ be the continuous functions from U to V, and $C(U)$ be the same as $C(U, \mathbb{R})$. We denote $C_b(U)$ the bounded continuous functions on U, and $C_0(U)$ or $C_c(U)$ be the functions in $C(U)$ with compact support. Then, $C^n = C^n(U)$ are the functions in $C(U, \mathbb{R})$ with continuous derivatives up to order n. Thus, $C_0^n = C_0^n(U)$ are the functions in C^n with compact support in U. We denote $C^{n+\alpha}$ the functions in C^n whose nth derivatives are Lipschitz continuous with exponent α. At last, $C^{1,2}(\mathbb{R} \times \mathbb{R}^n)$ are the functions $f(t, x) : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ which are C^1 with respect to $t \in \mathbb{R}$ and C^2 with respect to $x \in \mathbb{R}^n$.

The notation $[a, c)$ is used to indicate an interval from a to c that is inclusive of a but exclusive of c. The endpoint adjoining the square bracket is known as closed, while the endpoint adjoining the parenthesis is known as open. We can also use the notation $[a, c[$. Whenever infinity or negative infinity is used as an endpoint in the case of intervals on the real number line, it is always considered open and adjoined to a parenthesis. The endpoint can be closed when considering intervals on the extended real number line.
Motivated by the algebra of set, where the symbols $\cap$ and $\cup$ represent intersection and union, respectively, these symbols have been introduced in lattice theory where we have join and meet. The least upper bound of $a, b \in A$ is denoted by $a \cup b$ and called the join of elements a, b and the greatest lower bound of $a, b \in A$ is denoted by $a \cap b$ and called the meet of a, b.
In propositional calculus we have $\vee$ for disjunction (introduced by Russell) and we have $\wedge$ for conjunction.
The link between the two notations is with boolean algebra and its use as interpretation for the propositional calculus.
Given the binary relation $\leqslant$ in every Boolean algebra, we write $p \leqslant q$ in case $p \wedge q = p$, and we have that: for each p and q, the set $\{p, q\}$ has the supremum $p \vee q$ and the infimum $p \wedge q$.
The supremum is the minimum of the set of upper bounds, and the infimum the maximum of the set of lower bound.

23.2 Miscellaneous

See textbook by Larson et al. [2009].

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

23.2.1 Some functions

23.2.1.1 Some optimums

23.2.1.1.1 The maxima and minima The maxima and minima (the respective plurals of maximum and minimum) of a function, known collectively as extrema (the plural of extremum), are the largest and smallest value of the function, either within a given range (the local or relative extrema) or on the entire domain of a function (the global or absolute extrema).

Definition 23.2.1 *A real-valued function f defined on a domain X has a global (or absolute) maximum point at x^* if $f(x^*) \geqslant f(x)$ for all $x \in X$. Similarly, the function has a global (or absolute) minimum point at x^* if $f(x^*) \leqslant f(x)$ for all $x \in X$.*

If the domain X is a metric space then f is said to have a local (or relative) maximum point at the point x^* if there exists some $\epsilon > 0$ such that $f(x^*) \geqslant f(x)$ for all $x \in X$ within distance ϵ of x^*. Similarly, the function has a local (or relative) minimum point at the point x^* if there exists some $\epsilon > 0$ such that $f(x^*) \leqslant f(x)$ for all $x \in X$ within distance ϵ of x^*. A similar definition can be used when X is a topological space, since the definition just given can be rephrased in terms of neighbourhoods.
Finding global maxima and minima is the goal of mathematical optimisation. If a function is continuous on a closed interval, then by the extreme value theorem global maxima and minima exist. Furthermore, a global maximum (or minimum) either must be a local maximum (or minimum) in the interior of the domain, or must lie on the boundary of the domain. So a method of finding a global maximum (or minimum) is to look at all the local maxima (or minima) in the interior, and also look at the maxima (or minima) of the points on the boundary, and take the largest (or smallest) one. Local extrema of differentiable functions can be found by Fermat's theorem, which states that they must occur at critical points. One can distinguish whether a critical point is a local maximum or local minimum by using the first derivative test, second derivative test, or higher-order derivative test, given sufficient differentiability. For any function that is defined piecewise, one finds a maximum (or minimum) by finding the maximum (or minimum) of each piece separately, and then seeing which one is largest (or smallest).

23.2.1.1.2 The infimum and supremum In set theory, the maximum and minimum of a set are the greatest and least elements in the set, respectively. The infimum (abbreviated inf; plural infima) of a subset S of a partially ordered set T is the greatest element in T that is less than or equal to all elements of S, if such an element exists. Consequently, the term greatest lower bound (abbreviated as GLB) is also commonly used. Similarly, the supremum (abbreviated sup; plural suprema) of a subset S of a partially ordered set T is the least element in T that is greater than or equal to all elements of S, if such an element exists. Consequently, the supremum is also referred to as the least upper bound (or LUB).
The concepts of infimum and supremum are similar to minimum and maximum, but are more useful in analysis because they better characterize special sets which may have no minimum or maximum.

Definition 23.2.2 *A lower bound of a subset S of a partially ordered set $(P, \leqslant)$ is an element a of P such that $a \leqslant x$ for all $x \in S$. A lower bound a of S is called an infimum (or greatest lower bound, or meet) of S if for all lower bounds y of S in P, $y \leqslant a$ (a is larger than or equal to any other lower bound).*

Definition 23.2.3 *Similarly, an upper bound of a subset S of a partially ordered set $(P, \leqslant)$ is an element b of P such that $b \geqslant x$ for all $x \in S$. An upper bound b of S is called an supremum (or least upper bound, or join) of S if for all upper bounds z of S in P, $z \geqslant b$ (b is less than or equal to any other upper bound).*

Existence of an infimum of a subset S of P can fail if S has no lower bound at all, or if the set of lower bounds does not contain a maximal element. However, if an infimum or supremum does exist, it is unique.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

23.2.1.1.3 Argument of the maximum The arguments of the maxima (abbreviated arg max or argmax) are the points of the domain of some function at which the function values are maximised. In contrast to global maxima, referring to the largest outputs of a function, arg max refers to the inputs, or arguments, at which the function outputs are as large as possible.

Definition 23.2.4 *Given an arbitrary set X, a totally ordered set Y, and a function $f : X \to Y$, the arg max over some subset S of X is defined by*

$$\arg\max_{x \in S \subseteq X} f(x) = \{x|\, xS \wedge \forall y \in S\ f(y) \leqslant f(x)\}$$

Similarly, $\arg\min$ stands for argument of the minimum, and is given by

$$\arg\min_{x \in S \subseteq X} f(x) = \{x|\, xS \wedge \forall y \in S\ f(y) \geqslant f(x)\}$$

If $S = X$ or S is clear from the context, then S then S is often left out. Thus, $\arg\max$ is the set of points x for which $f(x)$ attains the function's largest value (if it exists).
The $\arg\max$ operator is to be distinguished from the $\max$ operator which, given the same function, returns the maximum value instead of the point or points that reach that value. That is,

$$\max_x f(x) \text{ is the elements in } \{f(x)|\ \forall y : f(y) \leqslant f(x)\}$$

If M is the maximum of f, then the $\arg\max$ is the level set of the maximum:

$$\arg\max_x f(x) = \{x|f(x) = M\} = f^{-1}(M)$$

Rearranging, we get the identity

$$f\big(\arg\max_x f(x)\big) = \max_x f(x)$$

23.2.1.2 The derivative

The essence of differentiation is finding the ratio between the difference in value of $f(x)$ and the increment in x. The derivative, or slope of a function is given by

$$f^{'}(x) = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Generally, if $f = f(g(x))$ is a differentiable function of g and g is a differentiable function of x, then

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$$

where df, dg, and dx are, respectively, the differentials of the functions f, g and x. This is the chain rule of differentiation which is formally stated as follow:

Theorem 23.2.1 *The chain rule of differentiation*
If g is differentiable in x_0 and if h is differentiable in $u_0 = g(x_0)$, then $h(x) = h(g(x))$ is differentiable in x_0 and

$$\frac{dh}{dx}\Big|_{x=x_0} = \frac{dh}{du}\Big|_{u_0=g(x_0)} \times \frac{dg}{dx}\Big|_{x=x_0}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The product rule says that if both $u = u(x)$ and $v = v(x)$ are differentiable functions of x, then

$$\frac{d(uv)}{dx} = v\frac{du}{dx} + u\frac{dv}{dx}$$

Multiply both sides by the differential dx, we obtain

$$d(uv) = vdu + udv$$

which is the product rule in terms of differentials.
For any differentiable function $F(x)$, $dF = F^{'}(x)dx$ (with $F^{'}(x) = \frac{dF}{dx}$). Note, $F(x)$ is one antiderivative of the function $f(x)$ means $F^{'}(x) = f(x)$.

Definition 23.2.5 *Let f be a function defined on some interval (c, d), and let a be some number in this interval. The derivative of f at a is the value of the limit*

$$f^{'}(a) = \lim_{x \to a} = \frac{f(x) - f(a)}{x - a}$$

f is said to be differentiable at a if the limit exists.

The function f is called differentiable on the interval (c, d) if it is differentiable at every point a in (c, d).

Theorem 23.2.2 *If a function f is differentiable at some a in its domain, then f is also continuous at a.*

Definition 23.2.6 *A function F is called an antiderivative of f on the interval $[a, b]$ if one has $F^{'}(x) = f(x)$ for all x with $a < x < b$.*

A piecewise function is a function defined by multiple sub-functions, each sub-function applying to a certain interval of the main function's domain, a sub-domain. Piecewise is actually a way of expressing the function, rather than a characteristic of the function itself, but with additional qualification, it can describe the nature of the function. For example, a piecewise polynomial function is a function that is a polynomial on each of its sub-domains, but possibly a different one on each. The word piecewise is also used to describe any property of a piecewise function that holds for each piece but not necessarily hold for the whole domain of the function. A function is piecewise differentiable or piecewise continuously differentiable if each piece is differentiable throughout its subdomain, even though the whole function may not be differentiable at the points between the pieces.
A piecewise function is continuous on a given interval if the following conditions are met:

- it is defined throughout that interval,
- its constituent functions are continuous on the corresponding intervals (subdomains),
- there is no discontinuity at each endpoint of the subdomains within that interval.

In convex analysis, the notion of a derivative may be replaced by that of the subderivative for piecewise functions.

23.2.1.3 The subgradient

The subderivative, subgradient, and subdifferential generalise the derivative to convex functions which are not necessarily differentiable. Subderivatives arise in convex analysis, the study of convex functions, often in connection to convex optimisation (see Clarke [1983]).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 23.2.7 *Subderivative*
A subderivative of a convex function $f : I \to \mathbb{R}$ at a point x_0 in the open interval I is a real number c such that

$$f(x) - f(x_0) \geqslant c(x - x_0)$$

for all x in I.

One can show that the set of subderivatives at x_0 for a convex function is a nonempty closed interval $[a, b]$, where a and b are the one-sided limits

$$\begin{aligned} a &= \lim_{x \to x_0^-} \frac{f(x) - f(x_0)}{x - x_0} \\ b &= \lim_{x \to x_0^+} \frac{f(x) - f(x_0)}{x - x_0} \end{aligned}$$

which are guaranteed to exist and satisfy $a \leqslant b$. The set $[a, b]$ of all subderivatives is called the subdifferential of the function f at x_0. Since f is convex, if its subdifferential at x_0 contains exactly one subderivative, then f is differentiable at x_0.
The concepts of subderivative and subdifferential can be generalised to functions of several variables.

Definition 23.2.8 *Subgradient*
If $f : U \to \mathbb{R}$ is a real-valued convex function defined on a convex open set in the Euclidean space $\mathbb{R}^n$, a vector v in that space is called a subgradient at a point x_0 in U if for any x in U one has

$$f(x) - f(x_0) \geqslant v \cdot (x - x_0)$$

The set of all subgradients at x_0 is called the subdifferential at x_0 and is denoted $\partial f(x_0)$. The subdifferential is always a nonempty convex compact set.
$\partial f(x_t)$ denotes a subgradient of f at x_t and x_t is the t-th iterate of x. If f is differentiable, then its only subgradient is the gradient vector ∇f itself.
These concepts generalize further to convex functions $f : U \to \mathbb{R}$ on a convex set in a locally convex space V. A functional v^* in the dual space V^* is called subgradient at x_0 in U if for any x in U

$$f(x) - f(x_0) \geqslant v^*(x - x_0)$$

23.2.1.4 The kernel functions

Suppose that we would like to learn a nonlinear classification rule which corresponds to a linear classification rule for the transformed data points $\phi(x_i)$. However, when the dimension of $\phi(\cdot)$ becomes very large we should not use an explicit transformation. Note, if $\phi(x)$ is all ordered monomials of degree d, then $\phi^\top(x)\phi(x^{'}) = (x^\top x^{'})^d$ and we can implicitly work in a higher dimensional space by using a different dot product. This is called the kernel trick and

$$\phi^\top(x)\phi(x^{'}) = k(x, x^{'})$$

is a kernel function. Note, we can do an implicit transformation into infinite dimensional feature spaces if we use the right kernel function. For example, the Gaussian kernel function (or radial basis function) is given by

$$k(x, x^{'}) = e^{-\frac{\|x - x^{'}\|_2^2}{2h^2}}$$

We might also want to know the types of kernel functions corresponding to inner products in higher dimensional spaces.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 23.2.3 *Given some input space X, in order for a kernel function $k : X \to X$ to correspond to an inner product, it must satisfy:*

1. $k(x, x^{'}) = k(x^{'}, x)$, $\forall x, x^{'} \in X$ *(Symmetry)*
2. *For* $\{x_1, x_2, ..., x_m\} \subset X$, *the matrix*

$$K = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_m) \\ \vdots & \ddots & \vdots \\ k(x_1, x_1) & \cdots & k(x_1, x_m) \end{bmatrix}$$

known as the Gram matrix (or kernel matrix) is positive semidefinite.

The second condition is equivalent to each of the following conditions:

- $\forall \alpha \in \mathbb{R}^m$, $\alpha^\top K \alpha \geqslant 0$.
- All the eigenvalues of K are non-negative.

Some of the properties of kernel functions are:

- Suppose k_1, k_2 are kernel functions, $\alpha, \beta \geqslant 0$. Then,

$$k(x, x^{'}) = \alpha k_1(x, x^{'}) + \beta k_2(x, x^{'})$$

  is also a kernel function.
- In particular, if $\alpha = \beta = 1$, we have that

$$k(x, x^{'}) = k_1(x, x^{'}) + k_2(x, x^{'})$$

  is a kernel function.

Some useful kernel functions are the polynomial and radial basis (Gaussian) kernel functions given as follows:

$$\begin{aligned} &\text{Polynomial (inhomogeneous) function } : k(x_i, x_j) = (x_i \cdot x_j + 1)^d \\ &\text{Polynomial (homogeneous) function } : k(x_i, x_j) = (x_i \cdot x_j)^d \\ &\text{Gaussian radial basis function } : k(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2} \\ &\text{Hyperbolic tangent function } : k(x_i, x_j) = \tanh(\kappa x_i \cdot x_j + c) \text{ for some } \kappa > 0 \text{ and } c < 0. \end{aligned}$$

where d is the degree of the polynomial function and $\gamma > 0$ is the constant of the radial basis function. The latter is sometime given by $\gamma = \frac{1}{2}\sigma^2$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

23.2.1.5 The Dirac function

The Dirac delta function, or δ function, is a generalised function, or distribution that was historically introduced by the physicist Paul Dirac for modelling the density of an idealized point mass or point charge, as a function that is equal to zero everywhere except for zero and whose integral over the entire real line is equal to one. Laurent Schwartz introduced the theory of distribution, formalising and validating mathematically these computations. This amounts, in particular, to defining Dirac delta function as a linear functional, which maps every function to its value at zero.
The derivative of the Heaviside function $\mathcal{H}(x)$ is zero for $x \neq 0$, and at $x = 0$ the derivative is undefined. We represent the derivative of the Heaviside function by the Dirac delta function $\delta(x)$. The delta function is zero for $x \neq 0$ and infinite at the point $x = 0$. Since the derivative of $\mathcal{H}(x)$ is undefined, $\delta(x)$ is not a function in the conventional sense of the word. While we can derive the properties of the delta function rigorously, we will focus on heuristic treatments. The Dirac delta function can be loosely thought of as a function on the real line which is zero everywhere except at the origin, where it is infinite. It can be written as follow:

$$\delta(x) = \begin{cases} 0 \text{ for } x \neq 0 \\ \infty \text{ for } x = 0 \end{cases}$$

and

$$\int_{-\infty}^{\infty} \delta(x)dx = 1 \tag{23.2.1}$$

The second property comes from the fact that $\delta(x)$ represents the derivative of $\mathcal{H}(x)$.
For a translation, the integral of the time-delayed Dirac function is given by:

$$\int_{-\infty}^{\infty} f(t)\delta(t-T)dt = f(T) \tag{23.2.2}$$

This is sometimes referred to as the sifting property or the sampling property. The delta function is said to sift out the value at $t = T$.
It follows that the effect of convolving a function f(t) with the time-delayed Dirac delta is to time-delay $f(t)$ by the same amount:

$$\begin{aligned} (f(t) * \delta(t-T)) &= \int_{-\infty}^{\infty} f(\tau)\delta(t-T-\tau)d\tau \\ &= \int_{-\infty}^{\infty} f(\tau)\delta(\tau-(t-T))d\tau \text{ since } \delta(-x) = \delta(x) \\ &= f(t-T) \end{aligned}$$

This holds under the precise condition that f is a tempered distribution. As a special case, for instance, we have the identity (understood in the distribution sense)

$$\int_{-\infty}^{\infty} \delta(\xi - x)\delta(x-\nu)dx = \delta(\xi - \nu)$$

The Kronecker delta (named after Leopold Kronecker) is a function of two variables, usually just positive integers. The function is 1 if the variables are equal, and 0 otherwise:

$$\delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}$$

where the Kronecker delta δ_{ij} is a piecewise function of variables i and j. It appears naturally in many areas of mathematics, physics and engineering, as a means of compactly expressing its definition above. The following equations are satisfied:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- $\sum_j \delta_{ij} a_j = a_i$
- $\sum_i \delta_{ij} a_i = a_j$
- $\sum_k \delta_{ik} \delta_{kj} = \delta_{ij}$

so that the matrix δ can be considered as an identity matrix. The Kronecker delta has the so-called sifting property that for $j \in \mathbb{Z}$

$$\sum_{i=-\infty}^{\infty} \delta_{ij} a_i = a_j$$

and if the integers are viewed as a measure space, endowed with the counting measure, then this property coincides with the defining property of the Dirac delta function

$$\int_{-\infty}^{\infty} \delta(y-x) f(y) dy = f(x)$$

and in fact Dirac's delta was named after the Kronecker delta because of this analogous property.

23.2.1.6 The multiset

A multiset is a modification of the concept of a set that, unlike a set, allows for multiple instances for each of its elements. The positive integer number of instances, given for each element is called the multiplicity of this element in the multiset. As a consequence, an infinite number of multisets exist, which contain only elements a and b, but vary by the multiplicity of their elements:

- The set $\{a, b\}$ contains only elements a and b, each having multiplicity 1, when $\{a, b\}$ is seen as a multiset.
- In the multiset $\{a, a, b\}$, the element a has multiplicity 2, and b has multiplicity 1.
- In the multiset $\{a, a, a, b, b, b\}$, a and b both have multiplicity 3.

These objects are all different, when viewed as multisets, although they are the same set, since they all consist of the same elements. As with sets, and in contrast to tuples, order does not matter in discriminating multisets, so that $\{a, a, b\}$ and $\{a, b, a\}$ denote the same multiset. To distinguish between sets and multisets, a notation that incorporates square brackets is sometimes used: the multiset $\{a, a, b\}$ can be denoted as $[a, a, b]$. The cardinality of a multiset is constructed by summing up the multiplicities of all its elements.

Definition 23.2.9 *A multiset may be formally defined as a 2-tuple (A, m) where A is the underlying set of the multiset, formed from its distinct elements, and $m : A \to \mathbb{N}_{\geqslant 1}$ is a function from A to the set of the positive integers, giving the multiplicity, that is, the number of occurrences, of the element a in the multiset as the number $m(a)$.*

If $A = \{a_1, ..., a_n\}$ is a finite set, the multiset (A, m) is often represented as $\{a_1^{m(a_1)}, ..., a_n^{m(a_n)}\}$. For example, the multiset $\{a, a, b\}$ can be written $\{a^2, b\}$. It is possible to extend the definition of a multiset by allowing multiplicities of individual elements to be infinite cardinals instead of natural numbers, but not all properties carry over to this generalisation.
Elements of a multiset are generally taken in a fixed set U, sometimes called a universe, which is typically the set of natural numbers. An element of U that does not belong to a given multiset is said to have a multiplicity 0 in this multiset.
The support of a multiset A in a universe U is the underlying set of the multiset. Using the multiplicity function m, it is characterised as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\text{support}(A) = \{x \in U | \, m_A(x) > 0\}$$

A multiset is finite if its support is finite, or, equivalently, if its cardinality

$$|A| = \sum_{x \in \text{support}(A)} m_A(x) = \sum_{x \in U} m_A(x)$$

is finite. The empty multiset is the unique multiset with an empty support (underlying set), and thus a cardinality 0. Multisets have become an important tool in database theory, which often uses the synonym bag. For instance, multisets are often used to implement relations in database systems. Multisets also play an important role in computer science.

23.2.1.7 The sequence

A sequence is an enumerated collection of objects in which repetitions are allowed. Similarly to a set, it contains members (also called elements, or terms). The number of elements (possibly infinite) is called the length of the sequence. Unlike a set, the same elements can appear multiple times at different positions in a sequence, and order matters. Formally, a sequence can be defined as a function whose domain is either the set of the natural numbers (for infinite sequences) or the set of the first n natural numbers (for a sequence of finite length n).

Definition 23.2.10 *We define a sequence to be a function whose domain is an interval of integers.*

The position of an element in a sequence is its rank or index, it is the natural number from which the element is the image. It depends on the context or a specific convention, if the first element has index 0 or 1. When a symbol has been chosen for denoting a sequence, the nth element of the sequence is denoted by this symbol with n as subscript. For example, the nth element of the Fibonacci sequence is generally denoted F_n.
In computing and computer science, finite sequences are sometimes called strings, words or lists, the different names commonly corresponding to different ways of representing them in computer memory. Infinite sequences are called streams. A sequence of a finite length n is also called an n-tuple. The empty sequence $()$ is included in most notions of sequence, but may be excluded depending on the context. Sequences are useful for studying functions, spaces, and other mathematical structures using the convergence properties of sequences. In particular, sequences are the basis for series, which are important in differential equations and analysis.
There are a number of ways for denoting a sequence, some of which are more useful for specific types of sequences. For instance, we can specify a sequence by listing the elements. For example, the first four odd numbers form the sequence $(1, 3, 5, 7)$. Other notations can be useful for sequences whose pattern cannot be easily guessed, or for sequences that do not have a pattern such as the digits of π. For instance, we can write down a general formula for computing the nth term as a function of n, enclose it in parentheses, and include a subscript indicating the range of values that n can take. For example, the sequence of even numbers could be written as $(2n)_{n\in\mathbb{N}}$. The variable n is called an index, and the set of values that it can take is called the index set. It is often useful to combine this notation with the technique of treating the elements of a sequence as variables. This yields expressions like $(a_n)_{n\in\mathbb{N}}$ which denotes a sequence whose nth element is given by the variable a_n. Note, we can also consider a sequence of sequences: $\left((a_{m,n})_{n\in\mathbb{N}}\right)_{m\in\mathbb{N}}$ denotes a sequence whose mth term is the sequence $(a_{m,n})_{n\in\mathbb{N}}$. An alternative to writing the domain of a sequence in the subscript is to indicate the range of values that the index can take by listing its highest and lowest legal values. For example, the notation $(a_n)_{n=1}^{\infty}$ is the same as the sequence $(a_n)_{n\in\mathbb{N}}$ and does not contain an additional term at infinity. We usually write (a_k) for an arbitrary sequence. Often, the index k is understood to run from 1 to ∞. Note, sequences are frequently indexed starting from zero.
Sequences whose elements are related to the previous elements in a straightforward way are often defined using recursion. This is in contrast to the definition of sequences of elements as functions of their positions. For instance, the Fibonacci sequence can be defined using a recursive rule along with two initial elements. The rule is that each element is the sum of the previous two elements, and the first two elements are 0 and 1. That is

$$a_n = a_{n-1} + a_{n-1} \text{ with } a_0 = 0 \text{ and } a_1 = 1$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Many sequences have the property that each element of a sequence can be computed from the previous element. In this case, there is some function f such that for all n, $a_{n+1} = f(a_n)$.
A subsequence of a given sequence is a sequence formed from the given sequence by deleting some of the elements without disturbing the relative positions of the remaining elements. Formally, a subsequence of the sequence $(a_n)_{n\in\mathbb{N}}$ is any sequence of the form $(a_{n_k})_{k\in\mathbb{N}}$ where $(n_k)_{k\in\mathbb{N}}$ is a strictly increasing sequence of positive integers. other types of sequences that are easy to define include:

- An integer sequence is a sequence whose terms are integers.
- A polynomial sequence is a sequence whose terms are polynomials.
- A positive integer sequence is sometimes called multiplicative, if $a_{mn} = a_m a_n$ or all pairs n, m such that n and m are coprime.
- A binary sequence is a sequence whose terms have one of two discrete values. For example, base 2 values $(0, 1, 1, 0, ...)$, a series of coin tosses (Heads/Tails) $H, T, H, H, T, ...$, the answers to a set of True or False questions $(T, F, T, T, ...)$, and so on.

In analysis, when talking about sequences, one will generally consider sequences of the form $(x_1, x_2, ...)$ or $(x_0, x_1, ...)$ which is to say, infinite sequences of elements indexed by natural numbers. The most elementary type of sequences are numerical ones, that is, sequences of real or complex numbers. This type can be generalized to sequences of elements of some vector space. In analysis, the vector spaces considered are often function spaces. Even more generally, one can study sequences with elements in some topological space.
Infinite sequences of digits (or characters) drawn from a finite alphabet are of particular interest in theoretical computer science. They are often referred to simply as sequences or streams, as opposed to finite strings. Infinite binary sequences, for instance, are infinite sequences of bits (characters drawn from the alphabet $\{0, 1\}$). The set $C = \{0, 1\}^\infty$ of all infinite binary sequences is sometimes called the Cantor space. An infinite binary sequence can represent a formal language (a set of strings) by setting the nth bit of the sequence to 1 if and only if the nth string (in shortlex order) is in the language. This representation is useful in the diagonalisation method for proofs.

23.2.1.7.1 Convergence An important property of a sequence is convergence. If a sequence converges, it converges to a particular value known as the limit. If a sequence converges to some limit, then it is convergent. A sequence that does not converge is divergent.

Definition 23.2.11 *A sequence of real numbers (a_n) converges to a real number L if, for all $\epsilon > 0$ there exists a natural number N such that for all $n \geqslant N$, we have*

$$|a_n - L| < \epsilon$$

If (a_n) is a sequence of complex numbers rather than a sequence of real numbers, this last formula can still be used to define convergence, with the provision that $|\cdot|$ denotes the complex modulus, that is, $|z| = \sqrt{z^* z}$. If (a_n) is a sequence of points in a metric space, then the formula can be used to define convergence, if the expression $|a_n - L|$ is replaced by the expression $dist(a_n, L)$, which denotes the distance between a_n and L.
If (a_n) and (b_n) are convergent sequences, then the following limits exist, and can be computed as follows

- $\lim_{n\to\infty}\big(a_n \pm b_n\big) = \lim_{n\to\infty} a_n \pm \lim_{n\to\infty} b_n$
- $\lim_{n\to\infty} ca_n = c\lim_{n\to\infty} a_n$ for all c.
- $\lim_{n\to\infty}\big(a_n b_n\big) = \big(\lim_{n\to\infty} a_n\big)\big(\lim_{n\to\infty} b_n\big)$
- $\lim_{n\to\infty} \frac{a_n}{b_b} = \frac{\lim_{n\to\infty} a_n}{\lim_{n\to\infty} b_n}$, provided that $\lim_{n\to\infty} b_n \neq 0$.
- $\lim_{n\to\infty} a_n^p = \big(\lim_{n\to\infty} a_n\big)^p$ for all $p > 0$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

23.2.1.7.2 The series A series is, informally speaking, a description of the operation of adding infinitely many quantities, one after the other, to a given starting quantity. That is, a series is the sum of the terms of a sequence. The study of series is a major part of calculus and its generalisation, mathematical analysis. Series are used in most areas of mathematics, even for studying finite structures (such as in combinatorics), through generating functions.
An infinite series or simply a series is an infinite sum, represented by an infinite expression of the form

$$a_0 + a_1 + a_2 + \cdots$$

where (a_n) is any ordered sequence of terms, such as numbers, functions, or anything else that can be added (an abelian group). This is an expression that is obtained from the list of terms $a_0, a_1, ...$ by laying them side by side, and conjoining them with the symbol $+$. A series may also be represented by using summation notation, such as $\sum_{n=0}^{\infty} a_n$. If an abelian group A of terms has a concept of limit (for example, if it is a metric space), then some series, the convergent series, can be interpreted as having a value in A, called the sum of the series. This includes the common cases from calculus in which the group is the field of real numbers or the field of complex numbers. Given a series $s = \sum_{n=0}^{\infty} a_n$ its kth partial sum is

$$s_k = \sum_{n=0}^{k} a_n$$

By definition, the series s converges to the limit L (or simply sums to L), if the sequence of its partial sums has a limit L. In this case, one usually writes $L = \sum_{n=0}^{\infty} a_n$. A series s is said to converge or to be convergent when the sequence (s_k) of partial sums has a finite limit. If the limit of s_k is infinite or does not exist, the series is said to diverge When the limit of partial sums exists, it is called the value (or sum) of the series

$$\sum_{n=0}^{\infty} a_n = \lim_{k\to\infty} s_k = \lim_{k\to\infty} \sum_{n=0}^{k} a_n$$

23.2.1.8 Fixed point

A fixed point of a function is an element of the function's domain that is mapped to itself by the function. That is, x is a fixed point of the function f if $f(x) = x$. Not all functions have fixed points. Points that come back to the same value after a finite number of iterations of the function are called periodic points. A fixed point is a periodic point with period equal to one.
A topological space X is said to have the fixed point property (briefly FPP) if for any continuous function $f : X \to X$, there exists $x \in X$ such that $f(x) = x$. The FPP is a topological invariant, it is preserved by any homeomorphism. The FPP is also preserved by any retraction.
We present two useful theorems of functional analysis.

Theorem 23.2.4 *Banach's fixed point theorem*
Let (X, d) be a complete metric space and let $\gamma \in [0, 1[$. If $\phi : X \to X$ is a γ-contracting mapping (for any $x, y \in X$, $d(\phi(x), \phi(y)) \leqslant \gamma d(x, y)$), then there exists a unique x_∞ such that $\phi(x_\infty) = x_\infty$. Furthermore, the sequence $(x_n)_{n\in\mathbb{N}}$ defined by the recursive relation $x_{n+1} = \phi(x_n)$ with an arbitrary $x_0 \in X$ converges to x_∞ and also verifies $d(x_\infty, x_n) = O(\gamma^n)$.

Theorem 23.2.5 *Complete metric space*
Let $\mathcal{S}$ be a set. The space $L^\infty(\mathcal{S}) = \{V : \mathcal{S} \to \mathbb{R}$ measurable and bounded $\}$ with uniform norm $\|\cdot\|_\infty$ is a complete metric space.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

23.2.2 Some facts on convex and concave analysis

Details can be found on text books on convex analysis (see Rockafellar [1995]) as well as on textbooks on mathematics for economics (see Wilson [2012]). We use relint (S) to refer to the relative interior of a convex set S, which is the set S minus all of the points on the relative boundary. We use closure (S) to refer to the closure of S, the smallest closed set containing all of the limit points of S. We use ∇ to denote a differential operator that indicates taking gradient in vector calculus. In the Cartesian coordinate system $\mathbb{R}^n$ with coordinates $(x_1, ..., x_n)$ and standard basis $(\widehat{e}_1, .., \widehat{e}_n)$, the gradient is defined in terms of partial derivative operators as

$$\nabla = (\frac{\partial}{\partial x_1}, .., \frac{\partial}{\partial x_n}) = \sum_{i=1}^{n} \widehat{e}_i \frac{\partial}{\partial x_i}$$

The gradient product rule is given by

$$\nabla(fg) = f\nabla g + g\nabla f$$

and the rules for dot products satisfy

$$\nabla(u.v) = (u.\nabla)v + (v.\nabla)u + u \times (\nabla \times v) + v \times (\nabla \times u)$$

where u and v are vectors. The directional derivative of a scalar field $f(x, y, z)$ in the direction $a(x, y, z) = a_x\widehat{x} + a_y\widehat{y} + a_z\widehat{z}$ is defined as

$$a \cdot \text{ grad } f = a_x \frac{\partial f}{\partial x} + a_y \frac{\partial f}{\partial y} + a_z \frac{\partial f}{\partial z} = (a \cdot \nabla)f$$

which gives the change of a field f in the direction of a. The Laplace operator is a scalar operator that can be applied to either vector or scalar fields; for Cartesian coordinate systems it is defined as

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} = \nabla.\nabla = \nabla^2 \tag{23.2.3}$$

The tensor derivative of a vector field v can be denoted simply as $\nabla \otimes v$ where $\otimes$ represents the dyadic product. This quantity is equivalent to the transpose of the Jacobian matrix of the vector field with respect to space.

23.2.2.1 Convex functions

If $x, y \in \mathbb{R}$ and $\alpha \in (0, 1)$, then $(1-\alpha)x+\alpha y$ is a convex combination of x and y. Geometrically, a convex combination of x and y is a point somewhere between x and y. A set $X \subset \mathbb{R}$ is convex if $x, y \in X$ implies $(1 - \alpha)x + \alpha y \in X$ for all $\alpha \in [0, 1]$.

Definition 23.2.12 *A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if any of the following conditions hold*

1. *for all $x, y \in \mathbb{R}^n$,*
$$\frac{1}{2}f(x) + \frac{1}{2}f(y) \geqslant f(\frac{x+y}{2})$$
2. *for all $x, y \in \mathbb{R}^n$ and $\alpha \in [0, 1]$,*
$$(1 - \alpha)f(x) + \alpha f(y) \geqslant f((1 - \alpha)x + \alpha y)$$
3. *for all random variables X, Jensen's inequality*
$$E[f(X)] \geqslant f(E[X]) \tag{23.2.4}$$
is satisfied. If f is strictly convex then the equality implies that $X = E[X]$ w.p. 1.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

***Proposition* 13** *A sufficient condition for f to be convex is that*

$$\nabla^2 f \geq 0$$

where ≥ 0 stands for positive semi-definitness, that is, $\nabla^2 f$ has non-negative eigenvalues.

***Proposition* 14** *If f is convex and differentiable, then for all $x_0, \delta \in \mathbb{R}^n$,*

$$f(x_0 + \delta) \geqslant f(x_0) + \delta.\nabla f(x_0)$$

Theorem 23.2.6 *If f has a second derivative which is non-negative (positive) everywhere, then f is convex (strictly convex).*

23.2.2.2 Concave functions

Definition 23.2.13 *A function $f : X \to \mathbb{R}$ is concave if for all $x, y \in X$ and $\alpha \in [0, 1]$,*

$$(1 - \alpha)f(x) + \alpha f(y) \leqslant f((1 - \alpha)x + \alpha y)$$

Definition 23.2.14 *A function $f : X \to \mathbb{R}$ is strictly concave if for all $x, y \in X$ with $x \neq y$ and $\alpha \in [0, 1]$,*

$$(1 - \alpha)f(x) + \alpha f(y) < f((1 - \alpha)x + \alpha y)$$

Geometrically, a function f is concave if the cord between any two points on the function lies everywhere on or below the function itself.

Consider a list of functions $f_i : X \to \mathbb{R}$ for $i = 1, .., n$ and a list of numbers $\alpha_1, .., \alpha_n$. The function $f = \sum_{i=1}^n \alpha_i f_i$ is called a linear combination of $f_1, .., f_n$. If each of the weights $\alpha_i \geqslant 0$, then f is a non-negative linear combination of $f_1, .., f_n$.

Theorem 23.2.7 *Suppose $f_1, .., f_n$ are concave functions and $(\alpha_1, .., \alpha_n) \geqslant 0$. Then, $f = \sum_{i=1}^n \alpha_i f_i$ is also a concave function. If at least one f_j is also strictly concave and $\alpha_j > 0$, then f is strictly concave.*

Even though a concave function need not be differentiable everywhere, the right and left hand derivatives always exist on the interior of the domain and $f^-(x) \geqslant f^+(x)$. As a result, f is both right and left continuous and therefore continuous. However, concave functions need not be continuous at the boundary.

For differentiable functions, the following theorem provides a simple necessary and sufficient conditions for concavity.

Theorem 23.2.8 *Suppose $f : X \to \mathbb{R}$ is differentiable.*

1. *f is concave if and only if for each $x, y \in X$ we have*

$$f(y) - f(x) \leqslant f^{'}(x)(y - x)$$

2. *f is strictly concave if and only if the inequality is strict for each $x \neq y$.*

Even if a function is not differentiable everywhere, it is concave if and only if for each $x \in int(X)$, there is an $a \in \mathbb{R}$ such that $f(y) - f(x) \leqslant a(y - x)$ for all $y \in X$. This is an example of a supporting hyperplane for one dimension.

Theorem (23.2.8) implies that the first derivative function of a concave function is non-increasing.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 23.2.9 *Suppose $f : X \to \mathbb{R}$ is differentiable.*

1. *f is concave if and only if $f^{'}$ is non-increasing.*
2. *f is strictly concave if and only if $f^{''}$ is strictly decreasing.*

Theorem 23.2.10 *Suppose $f : X \to \mathbb{R}$ is twice differentiable.*

1. *f is concave if and only if $f^{''} \leqslant 0$.*
2. *if $f^{''} < 0$, then f is strictly concave.*

Note, f strictly concave does not imply that $f^{''}(x) < 0$ for all x. An example of strictly concave function $f : \mathbb{R}_{++} \to \mathbb{R}$ is

1. $f(x) = \frac{x^{\alpha}}{\alpha}$ for $\alpha \neq 0$, $\alpha < 1$
2. $f(x) = \log x$
3. $f(x) = bx - ax^2$ where $a > 0$

The following lemma is an immediate consequence of the definition of concave and convex functions.

Lemma 23.2.1 *f is a (strictly) convex function if and only if $-f$ is a (strictly) concave function.*

23.2.2.3 Some approximations

1. Upper bound on the exponential function obtained by linearising e^x in 0

$$e^x \geqslant 1 + x$$

2. Lower bound on the logarithm function, derived by using the log operator on

$$\log(1 + x) \geqslant x$$

3. This inequality follows from the convexity of $f(z) = e^{\alpha z}$

$$e^{\alpha x} \leqslant 1 + (e^{\alpha} - 1)x \text{ for } x \in [0, 1]$$

4. Another inequality

$$-\log(1 - x) \leqslant x + x^2 \text{ for } x \in [0, \frac{1}{2}]$$

23.2.3 Some notes on convolution

23.2.3.1 Description

In functional analysis, convolution is a mathematical operation on two functions (f and g). It produces a third function, that is typically viewed as a modified version of one of the original functions, giving the integral of the pointwise multiplication of the two functions as a function of the amount that one of the original functions is translated. Some features of convolution are similar to cross-correlation. For real-valued functions, of a continuous or discrete variable, it only differs from the latter in that either $f(x)$ or $g(x)$ is reflected about the y-axis. Thus, it is a cross-correlation of $f(x)$ and $g(-x)$, or $f(-x)$ and $g(x)$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The convolution of f and g, written $f * g$, is defined as the integral of the product of the two functions after one is reversed and shifted. As such, it is a particular kind of integral transform:

$$\begin{aligned}(f * g)(x) &= \int_{-\infty}^{\infty} f(y)g(x-y)dy \\ &= \int_{-\infty}^{\infty} f(x-y)g(y)dy\end{aligned}$$

The first equation is equivalent to a weighted average of the function $f(y)$ with with a weight given by $g(\cdot - y)$. As x changes, the weighting function emphasises different parts of the input function.
For functions f, g supported only on $[0, \infty)$, the integration limits can be truncated, resulting in

$$(f * g)(x) = \int_0^x f(y)g(x-y)dy \text{ for } f, g : [0, \infty) \to \mathbb{R}$$

In that case, the Laplace transform is more appropriate than the Fourier transform and boundary terms become relevant.
When a function g_T is periodic, with period T, for function f such that $f * g_T$ exists, the convolution is also periodic and identical to

$$(f * g_T)(t) \equiv \int_{t_0}^{t_0+T} \Big[\sum_{k=-\infty}^{\infty} f(\tau + kT)\Big] g_T(t-\tau)d\tau$$

where t_0 is an arbitrary choice.
If f and g are compactly supported continuous functions, then their convolution exists, and is also compactly supported and continuous (Hormander [1983], Chapter 1). More generally, if either function (say f) is compactly supported and the other is locally integrable, then the convolution $f * g$ is well-defined and continuous.
Convolution of f and g is also well defined when both functions are locally square integrable on $\mathbb{R}$ and supported on an interval of the form $[a, \infty)$ (or both are supported on $(-\infty, a]$).
Under some circumstances, it is possible to define the convolution of a function with a distribution, or of two distributions. If f is a compactly supported function and g is a distribution, then $f * g$ is a smooth function defined by a distributional formula analogous to

$$\int_{\mathbb{R}^d} f(y)g(x-y)dy$$

More generally, it is possible to extend the definition of the convolution in a unique way so that the associative law

$$f * (g * \varphi) = (f * g) * \varphi$$

remains valid in the case where f is a distribution, and g a compactly supported distribution (Hormander [1983], chapter 4.2).
The convolution of any two Borel measures μ and ν of bounded variation is the measure λ defined by (see Rudin [1962])

$$\int_{\mathbb{R}^d} f(x)d\lambda(x) = \int_{\mathbb{R}^d \times \mathbb{R}^d} f(x+y)d\mu(x)d\nu(x)$$

which agrees with the convolution defined when μ and ν are regarded as distributions. The convolution of measures also satisfies the following version of Young's inequality

$$\|\mu * \nu\| \leqslant \|\mu\|\|\nu\|$$

where the norm is the total variation of a measure. Because the space of measures of bounded variation is a Banach space, convolution of measures can be treated with standard methods of functional analysis that may not apply for the convolution of distributions.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

23.2.3.2 Some properties

We now describe a few algebraic properties of convolution.
The convolution defines a product on the linear space of integrable functions. This product satisfies the following algebraic properties, which formally mean that the space of integrable functions with the product given by convolution is a commutative algebra without identity (Strichartz [1994], chapter 3.3). Other linear spaces of functions, such as the space of continuous functions of compact support, are closed under the convolution, and so also form commutative algebras.

- Commutativity: $f * g = g * f$
- Associativity: $f * (g * h) = (f * g) * h$
- Distributivity: $f * (g + h) = (f * g) + (f * h)$
- Multiplicative identity: No algebra of functions possesses an identity for the convolution. The lack of identity is typically not a major inconvenience, since most collections of functions on which the convolution is performed can be convolved with a delta distribution or, at the very least (as is the case of L_1), admit approximations to the identity. The linear space of compactly supported distributions does, however, admit an identity under the convolution. Specifically,
$$f * \delta = f$$
where δ is the Dirac delta function. It corresponds to Equation (23.2.2).
- Inverse element: Some distributions have an inverse element for the convolution, denoted S^{-1}, which is defined by
$$S^{-1} * S = \delta$$
The set of invertible distributions forms an abelian group under the convolution.
- Complex conjugation:
$$\overline{f * g} = \overline{f} * \overline{g}$$
- Relationship with differentiation:
$$(f * g)' = f' * g = f * g'$$
- Relationship with integration: If $F(x) = \int_{-\infty}^{x} f(y)dy$ and $G(x) = \int_{-\infty}^{x} g(y)dy$, then
$$(F * g)(x) = (f * G)(x) = \int_{-\infty}^{x} (f * g)(y)dy$$
- Integration: If f and g are integrable functions, then the integral of their convolution on the whole space is simply obtained as the product of their integrals:
$$\int_{\mathbb{R}^d} (f * g)(x)dx = \Big(\int_{\mathbb{R}^d} f(x)dx\Big)\Big(\int_{\mathbb{R}^d} g(x)dx\Big)$$
This follows from Fubini's theorem. The same result holds if f and g are only assumed to be nonnegative measurable functions, by Tonelli's theorem.
- Differentiation: In the one-variable case, we have
$$\frac{d}{dx}(f * g) = \frac{df}{dx} * g = f * \frac{dg}{dx} \tag{23.2.5}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

More generally, in the case of functions of several variables, an analogous formula holds with the partial derivative:

$$\frac{\partial}{\partial x_i}(f * g) = \frac{\partial f}{\partial x_i} * g = f * \frac{\partial g}{\partial x_i}$$

A particular consequence of this is that the convolution can be viewed as a smoothing operation: the convolution of f and g is differentiable as many times as f and g are in total.
These identities also hold much more broadly in the sense of tempered distributions if one of f or g is a compactly supported distribution or a Schwartz function and the other is a tempered distribution. On the other hand, two positive integrable and infinitely differentiable functions may have a nowhere continuous convolution.
In the discrete case, the difference operator $Df(n) = f(n+1) - f(n)$ satisfies an analogous relationship:

$$D(f * g) = (Df) * g = f * (Dg)$$

- Convolution theorem: The convolution theorem states that

$$\mathcal{F}\{f * g\} = k.\mathcal{F}\{f\}.\mathcal{F}\{g\}$$

where $\mathcal{F}\{f\}$ denotes the Fourier transform of f, and k is a constant that depends on the specific normalisation of the Fourier transform. Versions of this theorem also hold for the Laplace transform, two-sided Laplace transform, Z-transform and Mellin transform.

- Translation invariance: The convolution commutes with translations, meaning that

$$\tau_x(f * g) = (\tau_x f) * g = f * (\tau_x g)$$

where $\tau_x f$ is the translation of the function f by x defined by

$$(\tau_x f)(y) = f(y - x) \tag{23.2.6}$$

If f is a Schwartz function, then $\tau_x f$ is the convolution with a translated Dirac delta function

$$\tau_x f = f * \tau_x \delta$$

So translation invariance of the convolution of Schwartz functions is a consequence of the associativity of convolution. Furthermore, under certain conditions, convolution is the most general translation invariant operation. That is, suppose S is a linear operator acting on functions which commutes with translation

$$S(\tau_x f) = \tau_x(Sf) \text{ for all } x \tag{23.2.7}$$

Then S is given as convolution with a function (or distribution) g_S. That is,

$$Sf = g_S * f$$

Thus any translation invariant operation can be represented as a convolution. Convolutions play an important role in the study of time-invariant systems, and especially LTI system theory. The representing function g_S is the impulse response of the transformation S. Note, we must specify the class of functions on which the convolution is defined, and S must be a continuous linear operator with respect to the appropriate topology. It is known, for instance, that every continuous translation invariant continuous linear operator on L_1 is the convolution with a finite Borel measure. More generally, every continuous translation invariant continuous linear operator on L_p for $1 \leqslant p < \infty$ is the convolution with a tempered distribution whose Fourier transform is bounded.

23.2.3.3 Signal analysis

See textbook by Damelin et al. [2011].

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

23.2.3.3.1 Discrete convolution Discrete convolution is at the heart of any digital signal processing (DSP) system. This is illustrated in Figure (23.1).

Figure 23.1: DSP system

For complex-valued functions f, g defined on the set Z of integers, the discrete convolution of f and g is given by

$$\begin{aligned}(f*g)[n] &= \sum_{m=-\infty}^{\infty} f[m]g[n-m] \\ &= \sum_{m=-\infty}^{\infty} f[n-m]g[m] \quad \text{commutativity}\end{aligned}$$

In the above DSP system we get $y[n] = x[n] * h[n]$.
The convolution of two finite sequences is defined by extending the sequences to finitely supported functions on the set of integers. When the sequences are the coefficients of two polynomials, then the coefficients of the ordinary product of the two polynomials are the convolution of the original two sequences. This is known as the Cauchy product of the coefficients of the sequences. Thus, when g has finite support in the set $\{-M, -M+1, ..., M-1, M\}$ (representing, for instance, a finite impulse response), a finite summation may be used

$$(f*g)[n] = \sum_{m=-M}^{M} f[m]g[n-m]$$

The convolution of two complex-valued functions on $\mathbb{R}^d$ is itself a complex-valued function on $\mathbb{R}^d$ defined by

$$(f*g)(x) = \int_{\mathbb{R}^d} f(y)g(x-y)dy = \int_{\mathbb{R}^d} f(x-y)g(y)dy$$

It is well-defined only if f and g decay sufficiently rapidly at infinity in order for the integral to exist.

23.2.3.3.2 Correlation Correlation is an efficient way for comparing signals with each other and search for similarities. The mathematical formulation of correlation is the cross-correlation sequence

$$r_{xy}[l] = \sum_{n=-\infty}^{\infty} x[n]y[n-l]$$

For the time reversed cross-correlation sequence it can easily be shown that it is related to the original sequence as

$$r_{yx}[l] = \sum_{n=-\infty}^{\infty} y[n]x[n-l] = \sum_{m=-\infty}^{\infty} y[m+l]x[m] = r_{xy}[-l]$$

Autocorrelation is a cross-correlation sequence of a series with itself

$$r_{xx}[l] = \sum_{n=-\infty}^{\infty} x[n]x[n-l]$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, the zero lag term of the autocorrelation sequence is just the total energy of the signal

$$r_{xx}[0] = \sum_{n=-\infty}^{\infty} x^2[n] = \epsilon_x$$

Further, an autocorrelation sequence is even for real $x[n]$ since $r_{xx}[l] = r_{xx}[-l]$.

23.2.3.4 Multidimensional discrete convolution

See textbook by Dudgeon et al. [1983].

23.2.3.4.1 Description While convolution in one dimension allows the input and output of a linear shift-invariant (LSI) system to be easily compared so long as the impulse response of the filter system is known, this notion carries over to multidimensional convolution. In signal processing, multidimensional discrete convolution refers to the mathematical operation between two functions f and g on an n-dimensional lattice that produces a third function, also of n-dimensions. Multidimensional discrete convolution is the discrete analog of the multidimensional convolution of functions on Euclidean space. It is also a special case of convolution on groups when the group is the group of n-tuples of integers.
We represent an M-dimensional convolution of discrete signals as

$$y(n_1, n_2, ..., n_M) = x(n_1, n_2, ..., n_M) \overset{M}{*\cdots*} h(n_1, n_2, ..., n_M)$$

which can be computed as

$$\sum_{k_1}^{\infty}\sum_{k_2}^{\infty}\cdots\sum_{k_M}^{\infty} h(k_1, k_2, ..., k_M)x(n_1 - k_1, n_2 - k_2, ..., n_M - k_M)$$

The resulting output region of support of a discrete multidimensional convolution is determined based on the size and regions of support of the two input signals. We list below a few properties of the two-dimensional convolution operator which can be extended for N-dimensional signals.

- Commutative property: $x**h = h**x$
- Associative property: $(x**h)**g = x**(h**g)$
- Distributive property: $x**(h+g) = (x**h) + (x**g)$

23.2.3.4.2 Separable signals A signal is said to be separable if it can be written as the product of multiple one-dimensional signals. That is,

$$x(n_1, n_2, ..., n_M) = x(n_1)x(n_2)\cdots x(n_M)$$

Some readily recognisable separable signals include the unit step function and the delta-dirac impulse function:

- unit step function: $u(n_1, n_2, ..., n_M) = u(n_1)u(n_2)\cdots u(n_M)$
- delta-dirac impulse function: $\delta(n_1, n_2, ..., n_M) = \delta(n_1)\delta(n_2)\cdots\delta(n_M)$

Since convolution is a linear operation, the multidimensional convolution of separable signals can be expressed as the product of many one-dimensional convolutions. In he case where both x and h are separable functions, we get

$$x(n_1, n_2)**h(n_1, n_2) = \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty} h(k_1, k_2)x(n_1 - k_1, n_2 - k_2)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Applying the properties of separability, we get

$$x(n_1,n_2)**h(n_1,n_2)=\Big(\sum_{k_1=-\infty}^{\infty}h(k_1)x(n_1-k_1)\Big)\Big(\sum_{k_2=-\infty}^{\infty}h(k_2)x(n_2-k_2)\Big)$$

Thus, it reduces to the product of one-dimensional convolutions

$$x(n_1,n_2)**h(n_1,n_2)=\big[x(n_1)*h(n_1)\big]\big[x(n_2)*h(n_2)\big]$$

This result can be extended to the convolution of two separable M-dimensional signals

$$x(n_1,n_2,...,n_M)*\overset{M}{\cdots}*h(n_1,n_2,...,n_M)=\big[x(n_1)*h(n_1)\big]\big[x(n_2)*h(n_2)\big]\cdots\big[x(n_M)*h(n_M)\big]$$

which is n_M one-dimensional convolutions.
The row-column method can be applied when one of the signals in the convolution is separable. The method exploits the properties of separability in order to achieve a method of calculating the convolution of two multidimensional signals that is more computationally efficient than direct computation of each sample (given that one of the signals are separable). We let $h(n_1,n_2)$ be the separable signal. Then,

$$\begin{aligned}y(n_1,n_2) &= \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}h(k_1,k_2)x(n_1-k_1,n_2-k_2)\\ &= \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}h(k_1)h(k_2)x(n_1-k_1,n_2-k_2)\\ &= \sum_{k_1=-\infty}^{\infty}h(k_1)\Big[\sum_{k_2=-\infty}^{\infty}h(k_2)x(n_1-k_1,n_2-k_2)\Big]\end{aligned}$$

The resulting convolution can be effectively calculated by first performing the convolution operation on all of the rows of $x(n_1,n_2)$, and then on all of its columns. This approach can be further optimised by taking into account how memory is accessed within a computer processor.

23.2.3.4.3 Gaussian convolution One application of multidimensional convolution used within signal and image processing is the Gaussian convolution referring to convolving an input signal with the Gaussian distribution function. Assuming $\mu=0$, the 1-dimension Gaussian distribution sampled at discrete value is given by

$$G(n)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{n^2}{2\sigma^2}}$$

Assuming an M-dimensional signal where σ is constant and $\mu_i=0$, for $i=1,..,M$, the Gaussian distribution is given by

$$G(n_1,n_2,...,n_M)=\frac{1}{(2\pi)^{\frac{M}{2}}\sigma^M}e^{-\frac{\sum_{i=1}^{M}n_i^2}{2\sigma^2}}$$

The M-dimensional signal is separable, such that

$$G(n_1,n_2,...,n_M)=G(n_1)G(n_2)\cdots G(n_M)$$

As a result, the Gaussian convolution with discrete-valued 1d-signal satisfies

$$y(n)=x(n)*G(n)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

and, for a Md-signal, it satisfies

$$y(n_1, n_2, ..., n_M) = x(n_1, n_2, ..., n_M) \overset{M}{*\cdots*} G(n_1, n_2, ..., n_M)$$

Gaussian convolution can be approximated by implementing a finite impulse response (FIR) filter. For example, in a two-dimensional filter, the transfer function is given by

$$H(z_1, z_2) = \frac{1}{s(r_1, r_2)} \sum_{k_1=-r_1}^{r_1} \sum_{k_2=-r_2}^{r_2} G(k_1, k_2) z_1^{-k_1} z_2^{-k_2}$$

where

$$s(r_1, r_2) = \sum_{k_1=-r_1}^{r_1} \sum_{k_2=-r_2}^{r_2} G(k_1, k_2)$$

Choosing lower values for r_1 and r_2 will result in performing less computations, but will yield a less accurate approximation. On the other hand, choosing higher values for r_1 and r_2 will yield a more accurate approximation, but will require a greater number of computations.
Another method for approximating Gaussian convolution is via recursive passes through a box filter. For approximating one-dimensional convolution, this filter is given by

$$H(z) = \frac{1}{2r+1} \frac{z^r - z^{-r-1}}{1 - z^{-1}}$$

In general, a few recursive passes are required to obtain an accurate approximation. One can compute r as follows:

$$\sigma^2 = \frac{1}{12} K\big((2r+1)^2 - 1\big)$$

where K is the number of recursive passes through the filter. Since the Gaussian distribution is separable across different dimensions, it follows that recursive passes through one-dimensional filters (isolating each dimension separately) will thus yield an approximation of the multidimensional Gaussian convolution. Thus, the M-dimensional Gaussian convolution can be approximated with recursive passes through the following one-dimensional filters

$$H(z_i) = \frac{1}{2r_i + 1} \frac{z_i^{r_i} - z_i^{-r_i-1}}{1 - z_i^{-1}}, i = 1, ..., M$$

Gaussian convolutions are used extensively in signal and image processing. It is also commonly used in computer vision applications such as Scale-invariant feature transform (SIFT) feature detection.

23.3 Introduction to wavelet theory

Fitting financial time series is a difficult task as they are not stationary and contain noise. It is possible to get rid of this noise by using a filter. However, when denoising a signal we need to preserve the relevant multi-scaled information, which can be done with wavelets.
We present a short introduction to wavelets and provide some examples on denoising techniques.

23.3.1 Some definitions

Wavelet theory overcomes some problems linked to Fourier theory (such as stationarity) by generalising the decomposition to a time-frequency domain. The word wavelet (ondelette) were used by Morlet and Grossmann in the early 1980s. Started with the work of Haar in the early 20th century, Zweig discovered of the continuous wavelet transform in 1975, which were extended by Goupillaud, Grossmann and Morlet. Stromberg introduced discrete wavelets (1983)

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

and Daubechies presented the orthogonal wavelets with compact support (1988). Wavelets were first used in physics to characterise the Brownian motion and have been widely used in diverse tasks related to financial time series analysis.

Definition 23.3.1 *Wavelet*
A wavelet is a square integrable function $\psi_{s,\tau} : t \in \mathbb{R} \to \mathbb{R}$*, where* $(s,\tau) \in \mathbb{R}^{*+} \times \mathbb{R}$*, such that* $\psi_{s,\tau} \in L^2(\mathbb{R})$ *and* $\int_{-\infty}^{+\infty} \psi_{s,\tau}(t)dt = 0$.

***Remark* 23.3.1** *Wavelets derived from a mother wavelet* Ψ *in the following way:* $\forall t \in \mathbb{R}, \psi_{s,\tau}(t) = \frac{1}{\sqrt{s}}\Psi(\frac{t-\tau}{s})$*. In that sense, a mother wavelet generates a subgroup* Λ *of the affine transformation group of* $\mathbb{R}^n$.

There are many wavelet families, each of them gathering wavelets having common properties.

23.3.1.1 Continuous Wavelet

As in Fourier theory, both continuous and discrete wavelet transforms exist. We first define the continuous wavelet transform (CWT) of a signal f as follows:

$$\forall (s,\tau) \in \mathbb{R}^{*+} \times \mathbb{R} \qquad g(s,\tau) = \int_{-\infty}^{+\infty} f(t)\psi^*_{s,\tau}(t)dt$$

$*$ stands for the complex conjugate, τ is a translation factor and s is a dilatation factor. To recover f from g, the following equation is used:

$$\forall t \in \mathbb{R} \qquad f(t) = \frac{1}{C}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \frac{1}{|s|^2} g(s,\tau)\psi_{s,\tau}(t)dsd\tau$$

where $C = \int_{-\infty}^{+\infty} \frac{|\hat{\Psi}(x)|}{|x|}dx$, $\hat{\Psi}$ being the Fourier transform of the mother wavelet Ψ.

An example of such wavelet is the Mexican hat wavelet. The continuous wavelet transform defines a priori an infinite number of coefficients which cannot be used in practice. For that reason, we will only make use of discrete wavelet transform (DWT) introduced below.

23.3.1.2 Discrete Wavelet

Consider a discrete time set $\mathbb{T} = [\![0, T-1]\!]$. We define a discrete wavelet on this set as follows:

$$\forall t \in \mathbb{T} \qquad \psi_{m,n}(t) = s_0^{-m/2}\psi(s_0^{-m}t - n\tau_0)$$

where s_0 and τ_0 are constant. Then, the scaling and wavelet functions of DWT are defined as follows:

$$\phi(2^m t) = \sum_{i=1}^{n} h_{m+1}(i)\phi(2^{m+1}t - i)$$

$$\psi(2^m t) = \sum_{i=1}^{n} g_{m+1}(i)\phi(2^{m+1}t - i)$$

where $(h_{m+1}(i))_{i\in[\![1,n]\!]}$ and $(g_{m+1}(i))_{i\in[\![1,n]\!]}$ are respectively the coefficient sequences of ϕ and ψ. From these functions, a discrete signal x can be written as:

$$x(t) = \sum_{i=1}^{n} \lambda_{m-1}(i)\phi(2^{m-1}t - i) + \sum_{i=1}^{n} \gamma_{m-1}(i)\psi(2^{m-1}t - i)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

If $\{\psi_{m,n} : m, n \in \mathbb{Z}\}$ defines an orthonormal basis of $L^2(\mathbb{R})$, the following formula is used in order to recover the signal x.

$$\forall t \in \mathbb{T} \qquad x(t) = \sum_{m\in\mathbb{Z}} \sum_{n\in\mathbb{Z}} \langle x, \psi_{m,n} \rangle \cdot \psi_{m,n}(t)$$

***Remark* 23.3.2** *Orthonormality is an interesting property but not necessarily required for wavelets' construction. Compact support is also an useful property wavelets have as they can calibrate data efficiently, though it is not required neither.*

***Remark* 23.3.3** *Using a set of wavelets having the orthonormal property makes it easier to recover a discrete signal from its coefficients* $(\langle x, \psi_{m,n} \rangle)_{m,n}$. *This is the case of Haar wavelets for example.*

Wavelets take care of temporal characteristics of the regular factors described earlier as they are defined in a two dimensional domain: frequency and time resolution combinaison. Wavelets are indeed small waves located at different times and therefore give significant information about both time and frequency domains. This new degree of freedom provides the ability to capture trends in a local fashion.

23.3.2 Decomposition schemes

We introduce three decomposition schemes standing for different approaches of the wavelet transform introduced earlier.

23.3.2.1 Discrete Wavelet Transform

DWT is the most basic wavelet decomposition, Figure (23.2) shows how it can be done. H, L and H', L' respectively stands for high-pass, low-pass filters for wavelet decomposition and reconstitution. In the decomposition phase, L and H remove respectively the higher frequencies and the lower frequencies to give an approximation and a detail of signal x. The result is then downsampled by 2. The right part of the figure denotes the reserve process and is known as the reconstruction phase.

Figure 23.2: DWT Filter Scheme

More generally, we can repeat this process n times by a cascade algorithm which decomposes and downsamples the latter approximation. The following figure describes the process. A signal x can then be decomposed at level n in the following fashion:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$x(t) = A_n(t) + \sum_{k=1}^{n} D_k(t) \tag{23.3.8}$$

Figure 23.3: DWT Decomposition Tree

The main drawback of DWT is to be time-variant. For example, if we consider a periodic signal x with period τ, the DWT of x on $[\![t-\tau, t]\!]$ is in general not the DWT of x on $[\![t, t+\tau]\!]$ due to the downsampling. This property being desirable, researchers have developed a new type of wavelet transform called stationary wavelet transform (SWT) which guarantees time-invariance.

23.3.2.2 Stationary Wavelet Transform

In the DWT framework the downsampling meant choosing even or odd indexed elements at every decomposition step. However, in general, we could choose between even and odd at every decomposition which would lead to many possible combinaisons. The cornerstone of SWT is to average these denoised signals. We choose 0 to be coinciding with an even downsampling and 1 with an odd one.

Definition 23.3.2 *ϵ-decimated DWT*
Let N be a decomposition level of a signal x and $\epsilon = (\epsilon_1, ..., \epsilon_N)$ a sequence of integers such that $\forall n \in [\![1, N]\!]$, $\epsilon_n \in \{0, 1\}$. We say that the DWT $\chi = (A_N, D_N, .., D_1)$ is ϵ-decimated DWT if $\forall n \in [\![1, N]\!]$, ϵ_n corresponds to the downsampling choice of χ at time n.

It is possible to calculate all the ϵ-decimated DWTs for a given finite signal x by computing χ for every possible sequence ϵ. However, doing so is very computationally inefficient as 2^N DWT need to be done for a given decomposition level N. Another way to do this is to avoid downsampling and perform an upsampling on the original filters H and L.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 23.4: SWT calculation at level n

By following this process we end up with a time-invariant wavelet transform χ. In the next part, we describe a more general decomposition which is worth being investigated.

23.3.2.3 Wavelet Packet Transform

WPT can be seen as an extension of DWT. The latter decomposes only the approximation coefficients while WPT decomposes both the approximation and detail coefficients. This leads to more complex and somehow more flexible decomposition. Figure (23.5) describes its process:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 23.5: WPT Decomposition Tree

However, this process leads to 2^N coefficients for a given decomposition level N and thus can be more cumbersome in terms of storage and computation processing.

23.3.3 Denoising techniques

In this section we explain how DWT, SWT and WPT can be used to denoise a given signal. We say that a signal is noised when it suffers from unwanted or unknown modifications. Although we can have an intuitive understanding of noise, particularly in financial time series, there cannot be clear definition of it. Consequently, researchers often model it. A common model used in the research papers is the Gaussian white noise which has the characteristic to have the same power spectral density at every bandwidth frequency.

Definition 23.3.3 *Power spectral density*
Let x a signal and X its Fourier transform. We define the power spectral density (PSD) of the signal x as $\Gamma_x = |X|^2 * T$.

Definition 23.3.4 *Gaussian white noise*
A signal x is called a Gaussian white noise process if x is a stationary Gaussian process such that its mean $\mu_x = 0$ and its PSD is constant.

From theses definition we state our goal: from a noised signal x defined on a domain D and carrying a Gaussian white noise ϵ, we wish to find the true signal x^* such that:

$$\forall t \in D \qquad x(t) = x^*(t) + \epsilon(t) \tag{23.3.9}$$

In the previous section, we have introduced the concept of wavelet decomposition leading to a set of approximation and detail coefficients. As the noise in a signal is generally contained in the detail ones, a common procedure consists in setting the smallest coefficients to zero. Given a threshold $\lambda \in \mathbb{R}^+$, we sum up the many possibilities below.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

soft	hard
$x \to \begin{cases} x-\lambda & \text{if} \quad x > \lambda \\ 0 & \text{if} \quad \lvert x\rvert \leqslant \lambda \\ x+\lambda & \text{if} \quad x < -\lambda \end{cases}$	$x \to \begin{cases} 0 & \text{if} \quad \lvert x\rvert < \lambda \\ x & \text{otherwise} \end{cases}$
greater	**less**
$x \to \begin{cases} 0 & \text{if} \quad x < \lambda \\ x & \text{otherwise} \end{cases}$	$x \to \begin{cases} 0 & \text{if} \quad x > \lambda \\ x & \text{otherwise} \end{cases}$

Table 23.1: Threshold Function Definitions

Figure 23.6: Thresholding functions ($\lambda = 2$)

It is a well-known fact that soft thresholding provides smoother results when comparing to other ones. However, hard thresholding provides better edge preservation.

Algorithm (68) defines our filtering process for Discrete Wavelet Transform and has been used in our experiments to denoise a signal.
Filtering processes for other wavelet decomposition methods are very similar, the general idea being thresholding detail coefficients.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 68 DWT filtering process

Require: get noised signal x
Require: set wavelet name wlt
Require: set decomposition level n
Require: set thresholding function f and level λ

```
[A(n), D_n, D_{n-1}, .., D_1] ← decomposition(x, wlt, n)
for i = 1 to n do
    D_i ← f(D_i, λ)
end for
x* ← reconstruction([A, D], wlt)
return x*
```

23.4 Dynamical systems theory

Initiated by Poincare [1892-99] [1905-10], dynamical systems theory is an area of mathematics used to describe the behaviour of the complex dynamical systems, usually by employing differential equations or difference equations. When differential equations are employed, the theory is called continuous dynamical systems. From a physical point of view, continuous dynamical systems is a generalisation of classical mechanics, where the equations of motion are postulated directly and are not constrained to be Euler-Lagrange equations of a least action principle. When difference equations are employed, the theory is called discrete dynamical systems. When the time variable runs over a set that is discrete over some intervals and continuous over other intervals or is any arbitrary time-set such as a cantor set, one gets dynamic equations on time scales. Some situations may also be modelled by mixed operators, such as differential-difference equations.

23.4.1 Introduction to dynamical systems

23.4.1.1 An overview

In mathematics, a dynamical system is a system in which a function describes the time dependence of a point in a geometrical space. At any given time, a dynamical system has a state given by a tuple of real numbers (a vector) that can be represented by a point in an appropriate state space (a geometrical manifold). The evolution rule of the dynamical system is a function that describes what future states follow from the current state. Often the function is deterministic, that is, for a given time interval only one future state follows from the current state. However, some systems are stochastic, in that random events also affect the evolution of the state variables.
In physics, a dynamical system is described as a particle, or ensemble of particles, whose state varies over time and thus obeys differential equations involving time derivatives. In order to make a prediction about the future behaviour of the system, an analytical solution of such equations or their integration over time through computer simulation is realised.
Dynamical systems theory has applications to a wide variety of fields such as mathematics, physics, biology, chemistry, engineering, economics, and medicine (see Strogatz [1994]). Dynamical systems are a fundamental part of chaos theory, logistic map dynamics, bifurcation theory, the self-assembly and self-organization processes, and the edge of chaos concept.

In Newtonian mechanics, the evolution rule of dynamical systems is an implicit relation that gives the state of the system for only a short time into the future. The relation is either a differential equation, difference equation or other time scale. To determine the state for all future times requires iterating the relation many times, each advancing time a small step. The iteration procedure is referred to as solving the system or integrating the system. If the system can be solved, given an initial point, it is possible to determine all its future positions, a collection of points known as a trajectory or orbit.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

While for simple dynamical systems, knowing the trajectory is often sufficient, most dynamical systems are too complicated to be understood in terms of individual trajectories. The difficulties arise because:

- The systems studied may only be known approximately. The parameters of the system may not be known precisely, or terms may be missing from the equations, which bring into question the validity or relevance of numerical solutions. To address these questions several notions of stability have been introduced, such as Lyapunov stability or structural stability. The stability of the dynamical system implies that there is a class of models or initial conditions for which the trajectories would be equivalent. The operation for comparing orbits to establish their equivalence changes with the different notions of stability.
- The type of trajectory may be more important than one particular trajectory. Some trajectories may be periodic, whereas others may wander through many different states of the system. Applications often require enumerating these classes or maintaining the system within one class. Classifying all possible trajectories has led to the qualitative study of dynamical systems, that is, properties that do not change under coordinate changes. Linear dynamical systems and systems that have two numbers describing a state are examples of dynamical systems where the possible classes of orbits are understood.
- The behaviour of trajectories as a function of a parameter may be what is needed for an application. As a parameter is varied, the dynamical systems may have bifurcation points where the qualitative behaviour of the dynamical system changes. For example, it may go from having only periodic motions to apparently erratic behaviour, as in the transition to turbulence of a fluid.
- The trajectories of the system may appear erratic, as if random. In these cases it may be necessary to compute averages using one very long trajectory or many different trajectories. The averages are well defined for ergodic systems and a more detailed understanding has been worked out for hyperbolic systems. Understanding the probabilistic aspects of dynamical systems has helped establish the foundations of statistical mechanics and of chaos.

23.4.1.2 Description

A dynamical system is a manifold M called the phase (or state) space endowed with a family of smooth evolution functions ϕ^t that for any element of $t \in T$, the time, map a point of the phase space back into the phase space. The notion of smoothness changes with applications and the type of manifold. There are several choices for the set T. When T is taken to be the reals, the dynamical system is called a flow, and if T is restricted to the non-negative reals, then the dynamical system is a semi-flow. When T is taken to be the integers, it is a cascade or a map, and the restriction to the non-negative integers is a semi-cascade.
For example, the evolution function ϕ^t can be the solution of a differential equation of motion

$$\dot{x} = v(x)$$

This equation gives the time derivative, represented by the dot, of a trajectory $x(t)$ on the phase space starting at some point x_0. The vector field $v(x)$ is a smooth function that at every point of the phase space M provides the velocity vector of the dynamical system at that point. Given a smooth function ϕ^t, an autonomous vector field can be derived from it. There is no need for higher order derivatives in the equation, nor for time dependence in $v(x)$ because these can be eliminated by considering systems of higher dimensions.
Other types of differential equations can be used to define the evolution rule. For instance, the equation arising from the modelling of mechanical systems with complicated constraints is given by

$$G(x, \dot{x}) = 0$$

The differential equations determining the evolution function ϕ^t are often ordinary differential equations. In this case the phase space M is a finite dimensional manifold. Many of the concepts in dynamical systems can be extended to

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

infinite-dimensional manifolds, those that are locally Banach spaces in which case the differential equations are partial differential equations.

23.4.1.2.1 Linear systems Linear dynamical systems can be solved in terms of simple functions and the behaviour of all orbits classified. In a linear system the phase space is the N-dimensional Euclidean space, so that any point in the phase space can be represented by a vector with N numbers. The linear systems satisfy a superposition principle: if $u(t)$ and $w(t)$ satisfy the differential equation for the vector field (but not necessarily the initial condition), then so will $u(t) + w(t)$.
For a flow, the vector field $\phi(x)$ is an affine function of the position in the phase space, that is,

$$\dot{x} = \phi(x) = Ax + b \tag{23.4.10}$$

where A is a matrix, b a vector of numbers and x the position vector. The solution to this system can be found by using the superposition principle. The case $b \neq$ with $A = 0$ is just a straight line in the direction of b:

$$\phi^t(x_1) = x_1 + bt$$

When b is zero and $A \neq 0$ the origin is an equilibrium (or singular) point of the flow, that is, if $x_0 = 0$, then the orbit remains there. For other initial conditions, the equation of motion is given by the exponential of a matrix. For an initial point x_0, we get

$$\phi^t(x_0) = e^{tA}x_0$$

When $b = 0$, the eigenvalues of A determine the structure of the phase space. From the eigenvalues and the eigenvectors of A it is possible to determine if an initial point will converge or diverge to the equilibrium point at the origin. The distance between two different initial conditions in the case $A \neq 0$ will change exponentially in most cases, either converging exponentially fast towards a point, or diverging exponentially fast. Linear systems display sensitive dependence on initial conditions in the case of divergence. For non-linear systems this is one of the (necessary but not sufficient) conditions for chaotic behaviour.
A discrete-time, affine dynamical system has the form of a matrix difference equation:

$$x_{n+1} = Ax_n + b \tag{23.4.11}$$

with A a matrix and b a vector. The change of coordinates $x \to x + (1 - A)^{-1}b$ removes the term b from the equation. In the new coordinate system, the origin is a fixed point of the map and the solutions are of the linear system $A^n x_0$. The solutions for the map are no-longer curves, but points that hop in the phase space. The orbits are organised in curves, or fibers, which are collections of points that map into themselves under the action of the map. As in the continuous case, the eigenvalues and eigenvectors of A determine the structure of phase space.

23.4.1.2.2 Nonlinear systems Simple non-linear dynamical systems and even piecewise linear systems do not satisfy the superposition principle. They can exhibit a completely unpredictable behaviour, which might seem to be random, despite the fact that they are fundamentally deterministic. This behaviour is called chaos. Hyperbolic systems are precisely defined dynamical systems that exhibit the properties ascribed to chaotic systems. In such systems the tangent space perpendicular to a trajectory can be well separated into two parts: one with the points that converge towards the orbit (the stable manifold) and another of the points that diverge from the orbit (the unstable manifold). Dynamical systems theory and chaos theory is concerned with finding whether a system will settle down to a steady state in the long term. If so, we need to find the possible attractors. We also need to find if the long-term behaviour of the system depend on its initial condition.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

23.4.2 Ergodic theory

A central concern of ergodic theory is the behaviour of a dynamical system when it is allowed to run for a long time. The first result in this direction is the Poincare recurrence theorem, which claims that almost all points in any subset of the phase space eventually revisit the set. More precise information is provided by various ergodic theorems which assert that, under certain conditions, the time average of a function along the trajectories exists almost everywhere and is related to the space average. Two of the most important theorems are those of Birkhoff [1931] and von Neumann which assert the existence of a time average along each trajectory. For the special class of ergodic systems, this time average is the same for almost all initial points: statistically speaking, the system that evolves for a long time forgets its initial state. Stronger properties, such as mixing and equidistribution, have also been extensively studied.
Ergodic theory is often concerned with ergodic transformation.

Definition 23.4.1 *Let $T : X \to$ be a measure-preserving transformation on a measure space (X, Σ, μ), with $\mu(X) = 1$. Then T is ergodic if for every E in Σ with $T^{-1}(E) = E$, either $\mu(E) = 0$ or $\mu(E) = 1$.*

Theorem 23.4.1 *Let $T : X \to$ be a measure-preserving transformation on a measure space (X, Σ, μ) and suppose f is a μ-integrable function, that is, $f \in L_1(\mu)$. Then we define the following averages:*

- *Time average: This is defined as the average (if it exists) over iterations of T starting from some initial point x*

$$\widehat{f}(x) = \lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} f(T^k x)$$

- *Space average: If $\mu(X)$ is finite and nonzero, we can consider the space or phase average of f*

$$\overline{f} = \frac{1}{\mu(X)} \int f d\mu$$

In general the time average and space average may be different. But if the transformation is ergodic, and the measure is invariant, then the time average is equal to the space average almost everywhere. The equidistribution theorem is a special case of the ergodic theorem, dealing specifically with the distribution of probabilities on the unit interval.
The pointwise or strong ergodic theorem states that the limit in the definition of the time average of f exists for almost every x and that the (almost everywhere defined) limit function $\widehat{f}$ is integrable:

$$\widehat{f} \in L_1(\mu)$$

Further, $\widehat{f}$ id T-invariant, that is

$$\widehat{f} \circ T = \widehat{f}$$

holds almost everywhere, and if $\mu(X)$ is finite, then the normalisation is the same

$$\int \widehat{f} d\mu = \int f d\mu$$

In particular, if T is ergodic, then $\widehat{f}$ must be constant (almost everywhere), and one has

$$\overline{f} = \widehat{f}$$

almost everywhere. Joining the first to the last claim, and assuming that $\mu(X)$ is finite and nonzero, one has that

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} f(T^k x) = \frac{1}{\mu(X)} \int f d\mu$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

for almost all x. That is, for all x except for a set of measure zero. For an ergodic transformation, the time average equals the space average almost surely.
We now give a probabilistic formulation due to Birkhoff and Khinchin.

Theorem 23.4.2 *Birkhoff-Khinchin theorem*
Let f be measurable, $E(|f|) < \infty$, and T be a measure-preserving map. Then with probability one:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} f(T^k x) = E[f|\mathcal{C}](x)$$

where $E[f|\mathcal{C}]$ is the conditional expectation given the σ-algebra $\mathcal{C}$ of invariant sets of T.

***Corollary* 4** *Pointwise ergodic theorem*
In particular, if T is also ergodic, then $\mathcal{C}$ is the trivial σ-algebra, and thus with probability one:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} f(T^k x) = E[f](x)$$

The origins of ergodic theory lie in the mechanics of gases (large-scale effects of the molecular dynamics) where the key rationale is that the systems considered are in equilibrium. It is permissible under strict conditions of stationarity (see Grimmet et al. [1992]). While the literature on ergodic systems is concerned with deterministic dynamics, the basic question whether time averages may be replaced by ensemble averages is equally applicable to stochastic systems, such as Langevin equations or lotteries. The essence of ergodicity is the question whether the system when observed for a sufficiently long time t samples all states in its sample space in such a way that the relative frequencies $f(x,t)dx$ with which they are observed approach a unique (independent of initial conditions) probability $P(x)dx$

$$\lim_{t\to\infty} f(x,t) = P(x)$$

If this distribution does not exist or is not unique, then the time average

$$\overline{A} = \lim_{T\to\infty} \frac{1}{T} \int_0^T A(x(t))dt$$

of an observable A can not be computed as an ensemble average in Huygens' sense,

$$< A >= \int_x A(x)P(x)dx$$

23.4.3 State space models

See textbooks by Hinrichsen et al. [2005], Friedland [2005], among others.

23.4.3.1 An overview

In control engineering, a state-space representation is a mathematical model of a physical system as a set of input, output and state variables related by first-order differential equations or difference equations. State variables are variables whose values evolve through time in a way that depends on the values they have at any given time, and also depends on the externally imposed values of input variables. The values of the output variables depend on the values of the state variables. The state space is the Euclidean space in which the variables on the axes are the state variables. The state of the system can be represented as a vector within that space. In addition, if the dynamical system is linear, time-invariant, and finite-dimensional, then the differential and algebraic equations may be written in matrix form. The state-space representation (also known as the time-domain approach) provides a convenient and compact way of

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

modelling and analysing systems with multiple inputs and outputs.
Examples of state space models (SSMs) include linear Gaussian models or the Kalman filtering model (see Kalman [1960]), where observations and latent states are assumed to be Gaussian. Other SSMs are the hidden Markov models (HMMs) (see Baum et al. [1966]), where observations and latent states are assumed to be discrete.

The internal state variables are the smallest possible subset of system variables that can represent the entire state of the system at any given time. The minimum number of state variables required to represent a given system, n, is usually equal to the order of the system's defining differential equation. If the system is represented in transfer function form, the minimum number of state variables is equal to the order of the transfer function's denominator after it has been reduced to a proper fraction.

23.4.3.2 Linear models

The most general state-space representation of a linear system with p inputs, q outputs and n state variables is given by

$$\begin{aligned} \dot{x}(t) &= A(t)x(t) + B(t)u(t) \\ y(t) &= C(t)x(t) + D(t)u(t) \end{aligned} \tag{23.4.12}$$

where $x(t) \in \mathbb{R}^n$ is the state vector, $\dot{x}(t) = \frac{d}{dt}x(t)$, $y(t) \in \mathbb{R}^q$ is the output vector, $u(t) \in \mathbb{R}^p$ is the input (or control) vector, A, B, C, D are matrices. The explicit discrete time-variant is given by

$$\begin{aligned} x(k+1) &= A(k)x(k) + B(k)u(k) \\ y(k) &= C(k)x(k) + D(k)u(k) \end{aligned}$$

A common method for feedback is to multiply the output by a matrix K and setting this as the input to the system $u(y) = Ky(t)$. Since the values of K are unrestricted the values can easily be negated for negative feedback. The presence of a negative sign (the common notation) is merely a notational one and its absence has no impact on the end results. So, the continuous time-invariant system becomes

$$\begin{aligned} \dot{x}(t) &= Ax(t) + BKy(t) \\ y(t) &= Cx(t) + DKy(t) \end{aligned}$$

Solving the output equation for $y(t)$ and substituting in the state equation, we get

$$\begin{aligned} \dot{x}(t) &= \big(A + BK(I - DK)^{-1}C\big)x(t) \\ y(t) &= (I - DK)^{-1}Cx(t) \end{aligned}$$

Note, the eigenvalues of A can be controlled by setting K appropriately through eigendecomposition of $\big(A + BK(I - DK)^{-1}C\big)$. It assumes that the closed-loop system is controllable or that the unstable eigenvalues of A can be made stable through appropriate choice of K.
For a strictly proper system D equals zero. Another fairly common situation is when all states are outputs, that is, $y = x$ which yields $C = I$, the Identity matrix. This would then result in the simpler equations

$$\begin{aligned} \dot{x}(t) &= \big(A + BK\big)x(t) \\ y(t) &= x(t) \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In addition to feedback, an input $r(t)$ can be added such that

$$u(t) = -Ky(t) + r(t)$$

In that case, the continuous time-invariant system becomes

$$\begin{aligned} \dot{x}(t) &= Ax(t) - BKy(t) + Br(t) \\ y(t) &= Cx(t) - DKy(t) + Dr(t) \end{aligned}$$

Solving the output equation for $y(t)$ and substituting in the state equation, we get

$$\begin{aligned} \dot{x}(t) &= \big(A - BK(I + DK)^{-1}C\big)x(t) + B\big(I - K(I + DK)^{-1}D\big)r(t) \\ y(t) &= (I + DK)^{-1}Cx(t) + (I + DK)^{-1}Dr(t) \end{aligned}$$

23.4.3.3 Non-linear models

In the case of non-linear systems, the most general form of a state-space model can be written as two functions:

$$\begin{aligned} \dot{x}(t) &= f(t, x(t), u(t)) \\ y(t) &= h(t, x(t), u(t)) \end{aligned} \tag{23.4.13}$$

The first equation is the state equation and the latter is the output equation. If the function f is a linear combination of states and inputs then the equations can be written in matrix notation like above. The $u(t)$ argument to the functions can be dropped if the system is unforced (it has no inputs).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 24

Some notions of probabilities

We provide an introduction to probability which can be found in textbooks by Grimmett et al. [1992], Todorovic [1992], Oksendal [1998], Klenke [2013], Suhov et al. [2014], among others.

24.1 Introduction to random variables

24.1.1 Events as sets and their probabilities

24.1.1.1 Sets of events

The occurrence or non-occurrence of an event A depends upon the chain of circumstances involved, called an experiment or trial. The result of an experiment is called its outcome. When it is non-predictable, we can only list the collection of possible outcomes.

Definition 24.1.1 *The set of all possible outcomes of an experiment is called the sample space and is denoted Ω.*

Definition 24.1.2 *An event is a property which can be observed either to hold or not to hold after the experiment is done. In mathematical terms, an event is a subset of Ω.*

The complement of a subset A of Ω is denoted by A^c. When considering the events A and B, we will also be concerned with the events $A \cup B$, $A \cap B$ and A^c representing A or B, A and B and not A, respectively. Events A and B are called disjoint if their intersection is the empty set $\varnothing$ (called the impossible set). The set Ω is called the certain event. While events are subsets of Ω, all subsets of Ω are not events.

24.1.1.2 Probability measures

Given that we can repeat an experiment a large number N of times, keeping the initial conditions as equal as possible, we let A be some event which may or may not occur on each repetition. We are interested in the likelihoods of the occurrence of such an event when N increases and denote $P(A)$ the probability that A occurs on any particular trial. If A and B are two disjoint events, then

$$P(A \cup B) = P(A) + P(B)\,,\, P(\varnothing) = 0\,,\, P(\Omega) = 1$$

Thus, P should be countably additive, and one can define the desirable properties of a probability function P applied to the set of events as:

Definition 24.1.3 *Probability measure*
A probability measure P on $(\Omega, \mathcal{F})$ is a function $P : \mathcal{F} \to [0, 1]$ satisfying

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1. *$P(\varnothing) = 0$ and $P(\Omega) = 1$*
2. *if $A_1, A_2, ...,$ is a collection of disjoint members of $\mathcal{F}$, so that $A_i \cap A_j = \varnothing$ for all pairs i, j satisfying $i \neq j$, then*

$$P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$$

More generally, a measure μ assigns positive numbers to sets $A : \mu(A) \in \mathbb{R}$. For instance, if A is a subset of Euclidean space, then $\mu(A)$ is a length, area, or volume. If A is an event, then $\mu(A)$ is a probability of the event. If $\mathcal{X}$ is a space, then $\mu(\mathcal{X})$ measures the whole space. Measures are non-negative (signed measures need not be), zero on the empty set, and countably additive on disjoint sets. In general, it is not possible to assign a measure to every subset of an arbitrary space $\mathcal{X}$ in a way consistent with these axioms.
Hence, to fully specify a measure space we need one more piece of data, the set of subsets of $\mathcal{X}$ which are measurable. These form a σ-algebra: they contain $\mathcal{X}$ and are closed under taking complements and countable unions (and hence, by De Morgan's laws, also closed under countable intersections). If we replace the word "countable" with "finite" we obtain the weaker notion of Jordan content, and if we replace it with "arbitrary" we get limp hogwash.
The triple $(\Omega, \mathcal{F}, P)$ comprising a set Ω, a σ-field $\mathcal{F}$ of subsets of Ω, and a probability measure P on $(\Omega, \mathcal{F})$, is called a probability space.

24.1.1.3 Conditional events

We now introduce the notion of conditional events:

Definition 24.1.4 *If $P(B > 0)$ then the conditional probability that A occurs given that B occurs is defined to be*

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

A family $B_1, B_2, ..., B_n$ of events is called a partition of Ω if

$$B_i \cap B_j = \varnothing \text{ when } i \neq j \text{ and } \bigcup_{i=1}^{n} B_i = \Omega$$

Each elementary $\omega \in \Omega$ belongs to exactly one set in a partition of Ω.

Lemma 24.1.1 *For any events A and B*

$$P(A) = P(A|B)P(B) + P(A|B^c)P(B^c)$$

More generally, let $B_1, B_2, ..., B_n$ be a partition of Ω. Then

$$P(A) = \sum_{i=1}^{n} P(A|B_i)P(B_i)$$

In general, the occurrence of some event B changes the probability that another event A occurs. The original probability $P(A)$ is replaced by $P(A|B)$. When the probability remains unchanged, we say that A and B are independent.

Definition 24.1.5 *Events A and B are independent if*

$$P(A \cap B) = P(A)P(B)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

More generally, a family $\{A_i; i \in I\}$ is called independent if

$$P(\bigcap_{i \in J} A_i) = \prod_{i \in J} P(A_i)$$

for all finite subsets J of I.

24.1.1.4 Chain rule

In probability theory, the chain rule allows the calculation of any member of the joint distribution of a set of random variables using only conditional probabilities.
In the case of two random events A and B, the chain rule is given by

$$P(A \cap B) = P(A|B) \cdot P(B)$$

For more than two events $A_1, ..., A_n$, the chain rule extends to the formula

$$P(A_n \cap \cdots \cap A_1) = P(A_n | A_{n-1} \cap \cdots \cap A_1) \cdot P(A_{n-1} \cap \cdots \cap A_1)$$

By induction, it can be rewritten as

$$P(A_n \cap \cdots \cap A_1) = \prod_{i=1}^{n} P\big(A_i | \bigcap_{j=1}^{i-1} A_j\big) \tag{24.1.1}$$

24.1.2 Defining random variables

A random variable can take on one of a set of different values, each with an associated probability. Its value at a particular time is subject to random variation.

Definition 24.1.6 *A random variable is a function $X : \Omega \to \mathbb{R}$ with the property that $\{\omega \in \Omega : X(\omega) \leqslant x\} \in \mathcal{F}$ for each $x \in \mathbb{R}$.*

Definition 24.1.7 *The distribution function of a random variable X is the function $F : \mathbb{R} \to [0, 1]$ given by*

$$F(x) = P(X \leqslant x)$$

We let $(\Omega, \mathcal{F}, P)$ denote a given complete probability space. A random variable X is an $\mathcal{F}$-measurable function $X : \Omega \to \mathbb{R}^n$. Every random variable induces a probability measure μ_X on $\mathbb{R}^n$, defined by

$$\mu_X(B) = P(X^{-1}(B))$$

where μ_X is called the **distribution** of X.

24.1.2.0.1 The joint probability distribution Given n random variables $X_1, ..., X_n$ defined on a probability space $(\Omega, \mathcal{F}, P)$, the joint probability distribution for $X_1, ..., X_n$, denoted $P(X_1, ..., X_n)$, is a probability distribution that gives the probability that each of $X_1, ..., X_n$ falls in any particular range or discrete set of values specified for that variable. In the case of only two random variables, this is called a bivariate distribution, but the concept generalises to any number of random variables, giving a multivariate distribution.
The joint probability distribution can be expressed either in terms of

- a joint **cumulative distribution function** (cdf) or

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- in terms of a joint textbfprobability density function (pdf) (in the case of continuous variables), or, joint **probability mass function** (pmf) (in the case of discrete variables).

These, in turn, can be used to find two other types of distributions:

1. The marginal distribution giving the probabilities for any one of the variables with no reference to any specific ranges of values for the other variables.
2. The conditional probability distribution giving the probabilities for any subset of the variables conditional on particular values of the remaining variables.

24.1.2.1 Discrete random variables

Discrete random variables take on one of a discrete range of values.

Definition 24.1.8 *The random variable X is called discrete if it takes values in some countable subset $\{x_1, x_2, ..\}$, only, of $\mathbb{R}$.*

Its distribution function $F(x) = P(X \leqslant x)$ is a jump function.

Definition 24.1.9 *The probability mass function of a discrete random variable X is the function $f : \mathbb{R} \to [0, 1]$ given by $f(x) = P(X = x)$.*

The distribution and mass functions are related by

$$F(x) = \sum_{i:x_i \leqslant x} f(x_i)$$

Lemma 24.1.2 *The probability mass function $f : \mathbb{R} \to [0, 1]$ satisfies*

1. *$f(x) \neq 0$ if and only if x belongs to some countable set $\{x_1, x_2, ..\}$*
2. *$\sum_i f(x_i) = 1$*

We let X and Y be two discrete random variables defined on the same experiment. They are completely specified by their joint probability mass function (pmf).

Definition 24.1.10 *The joint mass function $f : \mathbb{R}^2 \to [0, 1]$ of X and Y is given by*

$$f(x, y) = P(X = x \text{ and } Y = y)$$

The joint probability mass function of two discrete random variables X, Y is

$$P(X = x \text{ and } Y = y) = P(Y = y|X = x) \cdot P(X = x) = P(X = x|Y = y) \cdot P(Y = y) \tag{24.1.2}$$

where $P(Y = y|X = x)$ is the probability of $Y = y$ given that $X = x$. We write $f_{X,Y}(x, y) = P(X = x, Y = y)$ when we need to stress the role of X and Y.
Discrete variables X and Y are independent if the numerical value of X does not affect the distribution of Y.

Lemma 24.1.3 *X and Y are independent if and only if*

$$f_{X,Y}(x, y) = f_X(x) f_Y(y) \text{ for all } x, y \in \mathbb{R}$$

More generally, X and Y are independent if and only if $f_{X,Y}(x, y)$ can be factorised as the product $g(x)h(y)$ of a function of x alone and a function of y alone.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 24.1.11 *Discrete variables X and Y are independent if the events $\{X = x\}$ and $\{Y = y\}$ are independent for all x and y.*

Theorem 24.1.1 *If X and Y are independent and $g, h : \mathbb{R} \to \mathbb{R}$, then $g(X)$ and $h(Y)$ are also independent.*

The generalisation of the two-variable case is the joint probability distribution of n discrete random variables $X_1, X_2, ..., X_n$ called the chain rule of probability, and it is given by

$$\begin{aligned} P(X_1 = x_1, ..., X_n = x_n) &= P(X_1 = x_1) \times P(X_2 = x_2|X_1 = x_1) \\ &\times P(X_3 = x_3|X_1 = x_1, X_2 = x_2) \\ &\cdots \\ &\times P(X_n = x_n|X_1 = x_1, X_2 = x_2, ..., X_{n-1} = x_{n-1}) \end{aligned} \tag{24.1.3}$$

From Equation (24.1.1), the joint probability distribution can be rewritten as

$$P(\bigcap_{i=1}^{n} X_i) = \prod_{i=1}^{n} P\big(X_i| \bigcap_{j=1}^{i-1} X_j\big) \tag{24.1.4}$$

Since these are probabilities, we have in the case of n variables

$$\sum_i \sum_j \cdots \sum_k P(X_1 = x_{1i}, X_2 = x_{2j}, ..., X_n = x_{nk}) = 1$$

More generally, a family $\{X_i; i \in I\}$ of discrete random variables is independent if the events $\{X_i = x_i\}$, $i \in I$ are independent for all possible choices of the set $\{x_i; i \in I\}$ of values of the X_i. Thus, $\{X_i; i \in I\}$ is an independent family if and only if

$$P(X_i = x_i \text{ for all } i \in J) = \prod_{i \in J} P(X_i = x_i)$$

for all sets $\{x_i; i \in I\}$ and for all finite subsets J of I.
Given two discrete random variables X and Y with joint pmf $f_{X,Y}(x, y)$, we can define their marginal pmfs $f_X(x)$ and $f_Y(y)$ by using the law of total probability.

Lemma 24.1.4 *The marginal probability mass function*
If the joint probability mass function (pmf) of discrete random variables X and Y is $f_{X,Y}(x, y)$, then the marginal pmf are given by

$$\begin{aligned} f_X(x) &= P(X = x) = \sum_{y \in Range(Y)} P(X = x, Y = y) = \sum_{y \in Range(Y)} f_{X,Y}(x, y) \\ f_Y(y) &= P(Y = y) = \sum_{x \in Range(X)} P(X = x, Y = y) = \sum_{x \in Range(X)} f_{X,Y}(x, y) \end{aligned}$$

Definition 24.1.12 *The conditional probability mass function*
Let X and Y be two discrete random variables. Then the conditional probability mass function of Y given $X = x$ is defined as

$$f_{Y|X}(y|x) = P(Y = y|X = x) = \frac{P(Y = y, X = x)}{P(X = x)} = \frac{f_{X,Y}(x, y)}{f_X(x)}$$

for $y \in Range(Y)$ and $x \in Range(X)$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Lemma 24.1.5 *If X and Y are discrete random variables, then for $x \in Range(X)$,*

$$\sum_{y \in Range(Y)} f_{Y|X}(y|x) = 1$$

Similarly, for $y \in Range(Y)$,

$$\sum_{x \in Range(X)} f_{Y|X}(y|x) = 1$$

Proof: From the definition of the conditional probability mass function and the marginal probability mass function, we get

$$\begin{aligned} \sum_{y \in Range(Y)} f_{Y|X}(y|x) &= \sum_{y \in Range(Y)} \frac{f_{X,Y}(x,y)}{f_X(x)} \\ &= \frac{f_X(x)}{f_X(x)} = 1 \end{aligned}$$

***Remark* 24.1.1** *Thus, $\{f_{Y|X}(y|x); y \in Range(Y)\}$ is a probability distribution on $Range(Y)$.*

Let $x_1, x_2, .., x_N$ be the numerical outcomes of N repetitions of some experiment. The average of these outcomes is

$$m = \frac{1}{N} \sum_i x_i$$

In advance of performing these experiments we can represent their outcomes by a sequence $X_1, X_2, .., X_N$ of random variables, and assume that these variables are discrete with a common mass function f. Then, roughly speaking, for each possible value x, about $Nf(x)$ of the X_i will take that value x. So, the average m is about

$$m \approx \frac{1}{N} \sum_x xNf(x) = \sum_x xf(x)$$

where the summation is over all possible values of the X_i. This average is the expectation or mean value of the underlying distribution with mass function f.

Definition 24.1.13 *The mean value or expectation of X with mass function f is defined to be*

$$E[X] = \sum_{x: f(x) > 0} xf(x)$$

whenever this sum is absolutely convergent.

Definition 24.1.14 *If k is a positive integer, then the kth moment m_k of X is*

$$m_k = E[X^k]$$

The kth central moment σ_k is

$$\sigma_k = E[(X - m_1)^k]$$

The two moments of most use are $m_1 = E[X]$ and $\sigma_2 = E[(X - E(X))^2]$ called the mean and variance of X. The expectation of two variables is as follows:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Lemma 24.1.6

$$E[g(X,Y)] = \sum_{x,y} g(x,y) f_{X,Y}(x,y)$$

The following lemma discuss when two mass functions agrees in probability.

Lemma 24.1.7 *If $|f_X(x) - f_Y(y)| = \epsilon$, there exits $f(x,y)$ such that $f(x) = f_X(x)$ and $f(y) = f_Y(y)$ and $f(x = y) = \epsilon$. Then we say that $f_X(x)$ agrees with $f_Y(y)$ with probability ϵ.*

24.1.2.2 Continuous random variables

Definition 24.1.15 *The random variable X is called continuous if its distribution function can be expressed as*

$$F(x) = \int_{-\infty}^{x} f(u)du \, , \, x \in \mathbb{R}$$

for some integrable function $f : \mathbb{R} \to [0, \infty)$.

If X and Y are two continuous random variables having a joint probability density function (pdf) $f_{X,Y}(x,y)$, then

$$F_{X,Y}(x,y) = \int_{-\infty}^{x} \int_{-\infty}^{y} f_{X,Y}(u,v)dudv \text{ for } x, y \in (-\infty, \infty)$$

Further, if $F_{X,Y}(x,y)$ is differentiable both in x and y, then the joint pdf $f_{X,Y}(x,y)$ is defined as the derivative of the joint cumulative distribution function

$$f_{X,Y}(x,y) = \frac{\partial^2}{\partial x \partial y} F_{X,Y}(x,y) = \lim_{\Delta x, \Delta y \to 0} \frac{P\big(x < X \leqslant x + \Delta x, \; y < Y \leqslant y + \Delta y\big)}{\Delta x \Delta y}$$

It leads to the chain rule

$$f_{X,Y}(x,y) = f_{Y|X}(y|x) \cdot f_X(x) = f_{X|Y}(x|y) \cdot f_Y(y)$$

Given the random variables $X_1, ..., X_n$, using the conditional probability, the joint probability distribution is given by

$$P(X_1, ..., X_n) = P(X_n | \, X_{n-1}, ..., X_1) \cdot P(X_{n-1}, ..., X_1)$$

which we can rewrite as

$$P(\bigcap_{i=1}^{n} X_i) = \prod_{i=1}^{n} P\big(X_i | \bigcap_{j=1}^{i-1} X_j\big)$$

We are now going to define independence of random variables.

Definition 24.1.16 *Random variables X and Y are independent if their joint distribution function factors into the product of their marginal distribution functions*

$$F_{X,Y}(x,y) = F_X(x) F_Y(y)$$

Theorem 24.1.2 *Suppose X and Y are jointly continuous random variables. X and Y are independent if and only if given any two densities for X and Y their product is the joint density for the pair (X,Y), that is,*

$$f_{X,Y}(x,y) = f_X(x) f_Y(y)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Independence requires that the set of points where the joint density is positive must be the Cartesian product of the set of points where the marginal densities are positive, that is, the set of points where $f_{X,Y}(x,y) > 0$ must be (possibly infinite) rectangles.

Definition 24.1.17 *Conditional distribution and mass function*
The conditional distribution function of Y given $X = x$, written $F_{Y|X}(.|x)$ is defined by

$$F_{Y|X}(y|x) = P(Y \leqslant y|X = x)$$

for any x such that $P(X = x) > 0$.
The conditional mass function of Y given $X = x$, written $f_{Y|X}(.|x)$ is defined by

$$f_{Y|X}(y|x) = P(Y = y|X = x)$$

As a result, the conditional density function of $Y|X$ can be written as

$$f_{Y|X}(y|x) = \frac{f_{X,Y}(x,y)}{f_X(x)}$$

if $f_X(x) > 0$ and 0 otherwise. $f_{X,Y}(x,y)$ is the joint mass function given in Definition (24.1.10). If X and Y are independent, then $f_{X,Y}(x,y) = f_X(x)f_Y(y)$ and we get $f_{Y|X}(y|x) = f_Y(y)$. Rearranging the above expression, we get

$$f_{X,Y}(x,y) = f_{Y|X}(y|x)f_X(x)$$

and integrating both sides over x we get

$$f_Y(y) = \int_{-\infty}^{\infty} f_{Y|X}(y|x)f_X(x)dx$$

which is an application of the law of total probability for the continuous case.
The expectation of a discrete variable X is $E[X] = \sum_x xP(X = x)$ which is an average of the possible values of X, each value being weighted by its probability. For continuous variables, expectations are defined as integrals.

Definition 24.1.18 *The expectation of a continuous random variable X with density function f is*

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx$$

whenever this integral exists.

We shall allow the existence of $\int g(x)dx$ only if $\int |g(x)|dx < \infty$. Note, the definition of the kth moment m_k in Appendix (24.1.2.1) applies to continuous random variables, but the moments of X may not exist since the integral

$$E[X^k] = \int x^k f(x)dx$$

may not converge.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

24.1.3 The characteristic function, moments and cumulants

24.1.3.1 Definitions

We start by recalling some definitions together with the properties of the characteristic functions. The characteristic function of a random variable is the Fourier transform of its distribution

Definition 24.1.19 *The characteristic function of a $\mathbb{R}^d$ random variable X is the function $\Phi_X : \mathbb{R}^d \to \mathbb{C}$ defined by*

$$\Phi_X(z) = E[e^{iz.X}] = \int_{\mathbb{R}^d} e^{iz.x} d\mu_X(x) \text{ for } z \in \mathbb{R}^d$$

where μ_X is the measure of X.

The characteristic function of a random variable completely characterises its law. Smoothness properties of Φ_X depend on the existence of moments of the random variable X which is related on how fast the distribution μ_X decays at infinity. If it exists, the n-th moment m_n of a random variable X on $\mathbb{R}$ is

$$m_n = E[X^n]$$

The first moment of X called the mean or expectation measures the central location of the distribution. Denoting the mean of X by μ_X, the nth central moment of X, if it exists, is defined as

$$m_n^c = E[(X - \mu_X)^n]$$

The second central moment σ_X^2 called the variance of X measures the variability of X. The third central moment measures the symmetry of X with respect to its mean, and the fourth central moment measures the tail behaviour of X. In statistics, skewness and kurtosis, respectively the normalised third and fourth central moments of X are used to summarise the extent of asymmetry and tail thickness. They are defined as follows:

$$S = \frac{m_3^c}{(m_2^c)^{\frac{3}{2}}} = E[\frac{(X - \mu_X)^3}{\sigma_X^3}] \text{ and } K = \frac{m_4^c}{(m_2^c)^2} = E[\frac{(X - \mu_X)^4}{\sigma_X^4}] \tag{24.1.5}$$

Since $K = 3$ for a normal distribution, the quantity $K - 3$ is called the excess kurtosis. The moments of a random variable are related to the derivatives at 0 of its characteristic function.

***Proposition* 15** *If $E[|X|^n] < \infty$ then Φ_X has n continuous derivatives at $z = 0$ and*

$$m_k = E[X^k] = \frac{1}{i^k} \frac{\partial^k \Phi_X}{\partial z^k}(0)$$

***Proposition* 16** *X possesses finite moments of all orders iff $z \to \Phi_X(z)$ is C^∞ at $z = 0$. Then the moments of X are related to the derivatives of Φ_X by*

$$m_n = E[X^n] = \frac{1}{i^n} \frac{\partial^n \Phi_X}{\partial z^n}(0)$$

If X_i with $i = 1, .., n$ are independent random variables, the characteristic function of $S_n = X_1 + ... + X_n$ is the product of characteristic functions of individual variables X_i

$$\Phi_{S_n}(z) = \prod_{i=1}^{n} \Phi_{X_i}(z) \tag{24.1.6}$$

We see that $\Phi_X(0) = 1$ and that the characteristic function Φ_X is continuous at $z = 0$ and $\Phi_X(z) \neq 0$ in the neighborhood of $z = 0$. It leads to the definition of the cumulant generating function or log characteristic function of X.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 24.1.20 *There exists a unique continuous function Ψ_X called the cumulant generating function defined around zero such that*

$$\Psi_X(0) = 0 \text{ and } \Phi_X(z) = e^{\Psi_X(z)}$$

The cumulants k_n of a probability distribution are a set of quantities providing an alternative to the moments of the distribution. It is defined via the cumulant-generating function $\Psi_X(z)$, which is the natural logarithm of the moment generating function

$$\Psi_X(z) = \ln \Phi_X(z)$$

The cumulants k_n are obtained from the power series expansion of the cumulant generating function

$$\Psi_X(z) = \sum_{n=1}^{\infty} k_n \frac{z^n}{n!}$$

so that the nth cumulant can be obtained by differentiating the above equation n-times and evaluating the result at zero

$$k_n = \Psi_X^{(n)}(z)\big|_{z=0}$$

24.1.3.2 The first two moments

We consider the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and let X and Y be two discrete random variables. We are going to present some properties of expectations involving joint distributions. For constants $a, b \in \mathbb{R}$, we have

$$E[aX + bY] = aE[X] + bE[Y]$$

Next, given the definition of the kth moment m_k and the kth central moment σ_k in Appendix (24.1.2.1) we get the following definition

Definition 24.1.21

$$\begin{aligned}
Cov(X,Y) &= E[XY] - E[X]E[Y] = E\big[(X - E[X])(Y - E[Y])\big] \\
E[XY] &= \sum_{x,y} xy f_{XY}(x,y) \\
Var(X) &= Cov(X,X) = E[X^2] - (E[X])^2 = E\big[(X - E[X])^2\big] \\
\rho(X,Y) &= \frac{Cov(X,Y)}{\big(Var(X)Var(Y)\big)^{\frac{1}{2}}} \text{ whenever } Var(X), Var(Y) \neq 0 \text{ and all these quantities exist}
\end{aligned}$$

The covariance measures whether, or not, $(X - E[X])$ and $(Y - E[Y])$ have the same sign. Further, the correlation is scale invariant

$$\rho(aX + b, cY + d) = \rho(X,Y)$$

For random variables X, Y, Z and constants $a, b, c, d \in \mathbb{R}$, then, from the above definition, we get

- $Cov(X, a) = 0$
- $Cov(aX + b, cY + d) = acCov(X,Y)$
- $Cov(X + Y, Z) = Cov(X,Z) + Cov(Y,Z)$
- $Cov(X,Y) = Cov(Y,X)$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 24.1.3 *For X and Y random variables, whenever the correlation $\rho(X,Y)$ exists, it must satisfy*

$$-1 \leqslant \rho(X,Y) \leqslant 1$$

The correlation $\rho(X,Y)$ is a measure of the strength and direction of the linear relationship between X and Y. Also,

- if X, Y have non-zero variance, then $\rho \in [-1,1]$.
- Y is a linearly increasing function of X if and only if $\rho(X,Y) = 1$.
- Y is a linearly decreasing function of X if and only if $\rho(X,Y) = -1$.

If X and Y are independent, then $\rho(X,Y) = 0$ and we must have $Cov(X,Y) = 0$ which leads to

$$E[XY] = E[X]E[Y]$$

***Remark* 24.1.2** *Independence of X and Y implies $Cov(X,Y) = 0$ but not vice versa. That is, some times $\rho(X,Y) = 0$ and X and Y are dependent.*

Moreover, if X and Y are independent, we get

$$Var(X+Y) = Var(X) + Var(Y)$$

Otherwise, if they are correlated, that is $\rho(X,Y) \neq 0$ we set $Z = X + Y$ and plug it back into the variance equation

$$\begin{aligned}
Var(Z) &= E[Z^2] - (E[Z])^2 = E[(X+Y)^2] - (E[X+Y])^2 \\
&= E[X^2 + Y^2 + 2XY] - \big(E[X] + E[Y]\big)^2 \\
&= E[X^2] + E[Y^2] + E[2XY] - (E[X])^2 - (E[Y])^2 - 2E[X]E[Y] \\
&= E[X^2] - (E[X])^2 + E[Y^2] - (E[Y])^2 + 2\big(E[XY] - E[X]E[Y]\big) \\
&= Var(X) + Var(Y) + 2Cov(X,Y)
\end{aligned}$$

Thus, we get

$$Var(aX \pm bY) = a^2 Var(X) + b^2 Var(Y) \pm 2abCov(X,Y)$$

More generally, for n random variables $X_1, ..., X_n$ the variance becomes

$$Var(\sum_{i=1}^{n} X_i) = \sum_{i,j=1}^{n} Cov(X_i, X_j) = \sum_{i=1}^{n} Var(X_i) + \sum_{i \neq j}^{n} Cov(X_i, X_j) \tag{24.1.7}$$

It leads to the variance of a linear combination

$$Var(\sum_{i=1}^{n} a_i X_i) = \sum_{i,j=1}^{n} a_i a_j Cov(X_i, X_j) = \sum_{i=1}^{n} a_i^2 Var(X_i) + 2 \sum_{1 \leqslant i < j \leqslant n} Cov(X_i, X_j)$$

24.1.4 Conditional moments

The related concept of conditional probability dates back from Laplace who calculated conditional distributions. Kolmogorov [1933] formalised it by using the Radon-Nikodym theorem. Halmos and Doob [1953] generalised the conditional expectation by using sub-sigma-algebras.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

24.1.4.1 Conditional expectation

Let A and B be two events defined on a probability space. Let $f_n(A)$ denote the number of times A occurs divided by n. As n gets large $f_n(A)$ should be close to $P(A)$, that is, $\lim_{n\to\infty} f_n(A) = P(A)$. When computing $P(A|B)$ we do not want to count the occurrences of $A \cap B^c$ since we know B has occurred. Hence, counting only the occurrences of A where B also occurs, this is $nf_n(A \cap B)$. Now the number of trials is the number of occurrences of B (all other trials are discarded as impossible since B has occurred). Therefore, the number of relevant trials is $nf_n(B)$. Consequently we should have

$$P(A|B) \approx \frac{nf_n(A \cap B)}{nf_n(B)} = \frac{f_n(A \cap B)}{f_n(B)}$$

and taking limits in n motivates the definition of the conditional probability which is

Definition 24.1.22 *Given A and B two events, if $P(B) > 0$ then the conditional probability that A occurs given that B occurs is*

$$P(A/B) = \frac{P(A \cap B)}{P(B)}$$

A family $B_1, B_2, ..., B_n$ of events is called a partition of Ω if

$$B_i \cap B_j = \oslash \text{ when } i \neq j \text{ and } \bigcup_{i=1}^{n} B_i = \Omega$$

Lemma 24.1.8 *(Partition Equation) For any events A and B*

$$P(A) = P(A|B)P(B) + P(A|B^c)P(B^c)$$

More generally, let $B_1, B_2, ..., B_n$ be a (finite or countable) partition of Ω. Then

$$P(A) = \sum_{i=1}^{n} P(A|B_i)P(B_i)$$

This may be set in the more general context of the conditional distribution of one variable Y given the value of another variable X (Discrete Case). Let X and Y be two random variables with Y taking values in $\mathbb{R}$ and with X taking only countably many values.

***Remark* 24.1.3** *Suppose we know that the event $\{X = j\}$ for some value j has occurred. The expectation of Y may change given this knowledge.*

Indeed, if $Q(\Lambda) = P(\Lambda|X = j)$, it makes more sense to calculate $E^Q[Y]$ than it does to calculate $E^P[Y]$.

Definition 24.1.23 *Let X have values in $(x_1, x_2, ..., x_n, ..)$ and Y be a random variable. Then if $P(X = x_j) > 0$ the conditional expectation of Y given $\{X = x_j\}$ is defined to be*

$$E[Y|X = x_j] = E^Q[Y]$$

where Q is the probability given by $Q(\Lambda) = P(\Lambda|X = j)$, provided $E^Q[|Y|] < \infty$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 24.1.4 *In the previous setting, and if further Y is countably valued with values $(y_1, y_2, ..., y_n, ..)$ and if $P(X = x_j) > 0$, then*

$$E[Y|X = x_j] = \sum_{k=1}^{\infty} y_k P(Y = y_k|X = x_j)$$

provided the series is absolutely convergent.

Proof:

$$E[Y|X = x_j] = E^Q[Y] = \sum_{k=1}^{\infty} y_k Q(Y = y_k) = \sum_{k=1}^{\infty} y_k P(Y = y_k|X = x_j)$$

Recall, $P(Y = y_k|X = x_j) = \frac{P(Y=y_k,X=x_j)}{P(X=x_j)}$ so that we get

$$E[Y|X = x_j] = \sum_{k=1}^{\infty} y_k \frac{P(Y = y_k, X = x_j)}{P(X = x_j)}$$

End of proof.

Next, still with X having at most a countable number of values, we wish to define the conditional expectation of any real valued r.v. Y given knowledge of the random variable X, rather than given only the event $\{X = x_j\}$. To this end we consider the function

$$f(x) = \begin{cases} E[Y|X = x] \text{ if } P(X = x) > 0 \\ \text{any arbitrary value } \text{ if } P(X = x) = 0 \end{cases}$$

Definition 24.1.24 *Let X be countably valued and let Y be a real valued random variable. The conditional expectation of Y given X is defined to be*

$$E[Y|X] = f(X)$$

where f is given above and provided that it is well defined.

Definition 24.1.25 *Let $\psi(x) = E[Y|X = x]$. Then $\psi(x)$ is called the conditional expectation of Y given X, written $E[Y|X]$.*

***Remark* 24.1.4** *Note, a conditional expectation is actually a random variable.*

Theorem 24.1.5 *The conditional expectation $\psi(X) = E[Y|X]$ satisfies*

$$E[\psi(X)] = E[Y]$$

Theorem 24.1.6 *Let $\mathcal{G}, \mathcal{H}$ be σ-algebra such that $\mathcal{G} \subset \mathcal{H}$. Then,*

$$E[X|\mathcal{G}] = E[E[X|\mathcal{H}]|\mathcal{G}]$$

Computing expectation by conditioning: Using this theorem, we can compute the marginal mean $E[Y]$ as

$$\begin{aligned} E[Y] &= \sum_x E[Y|X = x]P(X = x) \\ &= \sum_x \sum_y y f_{Y|X}(y|x) f_X(x) \end{aligned} \tag{24.1.8}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Thus, the marginal mean is the weighted average of the conditional means, with weights equal to the probability of being in the subgroup determined by the corresponding value of the conditioning variable. Further, from the chain rule of probability in Equation (24.1.3), we have $f_{X,Y}(x,y) = f_{Y|X}(y|x)f_X(x)$, which is a probability function. Hence, we can write the marginal mean as

$$E[Y] = \sum_x \sum_y y f_{X,Y}(x,y) \tag{24.1.9}$$

We can express the variance of the random variable Y in terms of conditional variance as

$$Var(Y) = E[Var(Y|X)] + Var(E[Y|X])$$

In the continuous case, we can compute $E[Y]$ as

$$E[Y] = \int_{-\infty}^{\infty} E[Y|X = x] f_X(x) dx$$

Theorem 24.1.7 *Let X and Y be random variables, and suppose that $E[Y^2] < \infty$. The best predictor of Y given X is the conditional expectation $E[Y|X]$.*

24.1.4.2 Conditional variance

Consider $E[Y|X]$ as a new random variable U as follows: Randomly pick an x from the distribution X so that the new r.v. U has the value $E[Y|X = x]$. For example, $Y = height$ and $X = sex$. Randomly pick a person from the population in question

$$U = \begin{cases} E[Y|X = female] \text{ if the person is female} \\ E[Y|X = male] \text{ if the person is male} \end{cases}$$

If U is a discrete r.v. the expected value of this new random variable is

$$E[U] = \sum_u P(u)u$$

and we get

$$E[E[Y|X]] = \text{ weighted average of conditional means } = E[Y]$$

The definition of the (population) (marginal) variance of a random variable Y is

$$Var(Y) = E[(Y - E[Y])^2]$$

Letting $\overline{Y} = E[Y]$ and expending the above term, we can rewrite the variance as

$$Var(Y) = E[Y^2 + (\overline{Y})^2 - 2Y\overline{Y}] = E[Y^2] + (\overline{Y})^2 - 2\overline{Y}E[Y] = E[Y^2] - (\overline{Y})^2$$

Similarly, if we are considering a conditional distribution $Y|X$, we define the conditional variance

$$Var(Y|X) = E\big[(Y - E[Y|X])^2|X\big]$$

Alternatively, we can write the conditional variance as

$$Var(Y|X) = E[Y^2|X] - (E[Y|X])^2 \tag{24.1.10}$$

Proof:
Letting $\psi(X) = E[Y|X]$ we can rewrite the conditional variance as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$Var(Y|X) = E[Y^2|X] + (\psi(X))^2 - 2\psi(X)E[Y|X] = E[Y^2|X] - (\psi(X))^2$$

As with $E[Y|X]$, we can consider $Var(Y|X)$ as a random variable. Hence, we can compute the expected value of the conditional variance as

$$E[Var(Y|X)] = E[E[Y^2|X]] - E[(E[Y|X])^2]$$

Since the expected value of the conditional expectation of a random variable is the expected value of the original random variable the above equation simplifies to

$$E[Var(Y|X)] = E[Y^2] - E[(E[Y|X])^2] \tag{24.1.11}$$

We can also compute the variance of the conditional expectation as

$$Var(E[Y|X]) = E[(E[Y|X])^2] - (E[E[Y|X]])^2$$

and since $E[E[Y|X]] = E[Y]$ it simplifies to

$$Var(E[Y|X]) = E[(E[Y|X])^2] - (E[Y])^2 \tag{24.1.12}$$

Combining Equation (24.1.11) with Equation (24.1.12) we get

$$E[Var(Y|X)] + Var(E[Y|X]) = E[Y^2] - (E[Y])^2$$

which is the marginal variance $Var(Y)$. That is, the marginal variance is

$$Var(Y) = E[Var(Y|X)] + Var(E[Y|X]) \tag{24.1.13}$$

The marginal (overall) variance is the sum of the expected value of the conditional variance and the variance of the conditional means. Since variances are always non-negative, we get

$$Var(Y) \geqslant E[Var(Y|X)]$$

Further, since $Var(Y|X) \geqslant 0$ and $E[Var(Y|X)]$ must also be positive then

$$Var(Y) \geqslant Var(E[Y|X])$$

Note, $E[Var(Y|X)]$ is a weighted average of $Var(Y|X)$. We can also write the variance of the conditional expectation as

$$Var(E[Y|X]) = E\big[(E[Y|X] - E[Y])^2\big]$$

which is a weighted average of $(E[Y|X] - E[Y])^2$.

24.1.5 Multiple random variables

Following Appendix (24.1.2), we define the joint **cumulative distribution function** and the joint **probability density function** for n random variables.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

24.1.5.1 Discrete variables

Let $X = (X_1, X_2, ..., X_n)$ be an n-dimensional vector of random variables and A_k be a one dimensional event involving X_k. Events with product form are defined as

$$A = \{X_1 \in A_1\} \cap \{X_2 \in A_2\} \cap \cdots \cap \{X_n \in A_n\}$$

Then,

$$\begin{aligned} P(A) &= P(\{X_1 \in A_1\} \cap \{X_2 \in A_2\} \cap \cdots \cap \{X_n \in A_n\}) \\ &= P(X_1 \in A_1, ..., X_n \in A_n) \end{aligned}$$

These random variables are independent if

$$P(X_1 \in A_1, ..., X_n \in A_n) = P(X_1 \in A_1) \cdots P(X_n \in A_n)$$

The joint cumulative distribution function (cdf) of $X_1, X_2, ..., X_n$ is

$$F_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n) = P[X_1 \leqslant x_1, X_2 \leqslant x_2, ..., X_n \leqslant x_n]$$

In general, the variables $X_1, X_2, ..., X_n$ are completely specified by their joint cdf. The joint cdf of $X_1, X_2, ..., X_{n-1}$ is

$$F_{X_1,X_2,...,X_{n-1}}(x_1, x_2, ..., x_{n-1}) = F_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_{n-1}, \infty)$$

The joint probability mass function (pmf) (see Definition (24.1.10)) of n discrete random variables is

$$f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n) = P[X_1 = x_1, X_2 = x_2, ..., X_n = x_n]$$

The probability of event A is

$$P((X_1, ..., X_n) \in A) = \sum \cdots \sum_{x \in A} f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n)$$

where $x = (x_1, x_2, ..., x_n)$.
The marginal pmf (see Lemma (24.1.4)) for X_j is

$$f_{X_j}(x_j) = \sum_{x_1} \cdots \sum_{x_{j-1}} \sum_{x_{j+1}} \cdots \sum_{x_n} f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n)$$

The marginal pmf for $X_1, X_2, ..., X_{n-1}$ is

$$f_{X_1,X_2,...,X_{n-1}}(x_1, x_2, ..., x_{n-1}) = \sum_{x_n} f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n)$$

The conditional pmf (see Definition (24.1.12)) is

$$f_{X_n|X_1,...,X_{n-1}}(x_n|x_1, ..., x_{n-1}) = \frac{f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n)}{f_{X_1,X_2,...,X_{n-1}}(x_1, x_2, ..., x_{n-1})} \tag{24.1.14}$$

Note, it is sometime denoted as $f_{X_n}(x_n|x_1, ..., x_{n-1})$.
The conditional pmf of $(X_{k+1}, X_{k+2}, ..., X_n)$ given $(X_1, X_2, ..., X_k)$ is

$$f_{X_{k+1},...,X_n|X_1,...,X_k}(x_{k+1}, ..., x_n|x_1, ..., x_k) = \frac{f_{X_1,...,X_n}(x_1, ..., x_n)}{f_{X_1,...,X_k}(x_1, ..., x_k)} \tag{24.1.15}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The chain rule of probability (see Equation (24.1.3)) is

$$\begin{aligned} f_{X_1,X_2,...,X_n}(x_1,x_2,...,x_n) &= f_{X_n|X_1,...,X_{n-1}}(x_n|x_1,...,x_{n-1}) \\ &\times f_{X_{n-1}|X_1,...,X_{n-2}}(x_{n-1}|x_1,...,x_{n-2})\cdots f_{X_2|X_1}(x_2|x_1)f_{X_1}(x_1) \end{aligned}$$

So, we can rewrite the conditional pmf of $(X_{k+1},X_{k+2},...,X_n)$ given $(X_1,X_2,...,X_k)$ in Equation (24.1.15) as

$$f_{X_{k+1},...,X_n|X_1,...,X_k}(x_{k+1},...,x_n|x_1,...,x_k) = f_{X_{k+1}|X_1,...,X_k}(x_{k+1}|x_1,...,x_k)\cdots f_{X_n|X_1,...,X_{n-1}}(x_n|x_1,...,x_{n-1})$$

It can be expressed in the product form as

$$f_{X_{k+1},...,X_n|X_1,...,X_k}(x_{k+1},...,x_n|x_1,...,x_k) = \prod_{i=k}^{n-1} f_{X_{i+1}|X_{\leqslant i}}(x_{i+1}|x_{\leqslant i}) \tag{24.1.16}$$

where $f_{X_{i+1}|X_{\leqslant i}}(x_{i+1}|x_{\leqslant i}) = P(X_{i+1}=x_{i+1}|\ X_{\leqslant i}=x_{\leqslant i})$ and $x_{\leqslant i}=x_1,x_2,...,x_i$.

24.1.5.2 Continuous variables

The random variables $X_1,X_2,...,X_n$ are jointly continuous random variables if the probability of any n-dimensional event A is given by an n-dimensional integral of a probability density function

$$P[(X_1,...,X_n)\in A] = \int\cdots\int_{x\in A} f_{X_1,...,X_n}(x_1^{'},...,x_n^{'})dx_1^{'}...dx_n^{'}$$

where $f_{X_1,...,X_n}(x_1^{'},...,x_n^{'})$ is the joint probability density function (pdf).
The joint cdf of X is obtained from the joint pdf:

$$F_{X_1,...,X_n}(x_1,...,x_n) = \int_{-\infty}^{x_1}\cdots\int_{-\infty}^{x_n} f_{X_1,...,X_n}(x_1^{'},...,x_n^{'})dx_1^{'}...dx_n^{'}$$

The joint pdf (if the derivatives exist) is given by

$$f_{X_1,X_2,...,X_n}(x_1,x_2,...,x_n) = \frac{\partial^n}{\partial x_1\cdots\partial x_n}F_{X_1,X_2,...,X_n}(x_1,x_2,...,x_n)$$

The marginal pdf for a subset of random variables is obtained by integrating the other variables out. For instance, the marginal pdf of X_1 is

$$f_{X_1}(x_1) = \int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} f_{X_1,X_2,...,X_n}(x_1,x_2^{'},...,x_n^{'})dx_2^{'}...dx_n^{'}$$

The marginal pdf for $X_1,...,X_{n-1}$ is given by

$$f_{X_1,...,X_{n-1}}(x_1,...,x_{n-1}) = \int_{-\infty}^{\infty} f_{X_1,...,X_n}(x_1,...,x_{n-1},x_n^{'})dx_n^{'}$$

The conditional pdf is

$$f_{X_n}(x_n|x_1,...,x_{n-1}) = \frac{f_{X_1,X_2,...,X_n}(x_1,x_2,...,x_n)}{f_{X_1,X_2,...,X_{n-1}}(x_1,x_2,...,x_{n-1})}$$

The chain rule of probability is

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} f_{X_1,X_2,...,X_n}(x_1,x_2,...,x_n) &= f_{X_n}(x_n|x_1,...,x_{n-1}) \\ &\times f_{X_{n-1}}(x_{n-1}|x_1,...,x_{n-2})\cdots f_{X_2}(x_2|x_1)f_{X_1}(x_1) \end{aligned}$$

The variables $X_1,...,X_n$ are independent if

$$P[X_1 \in A_1,...,X_n \in A_n] = P[X_1 \in A_1]\cdots P[X_n \in A_n]$$

The variables $X_1,...,X_n$ are independent if and only if

$$F_{X_1,...,X_n}(x_1,...,x_n) = F_{X_1}(x_1)\cdots F_{X_n}(x_n)\,,\forall x_1,...,x_n$$

In the case of discrete random variables, the above equation becomes

$$f_{X_1,...,X_n}(x_1,...,x_n) = f_{X_1}(x_1)\cdots f_{X_n}(x_n)\,,\forall x_1,...,x_n$$

It can be written in product form as

$$f_{X_1,...,X_n}(x_1,...,x_n) = \prod_{i=1}^{n} f_{X_i}(x_i)\,,\forall x_1,...,x_n$$

An important special case is independent and identically distributed (i.i.d.) random variables. The variables $X_1, X_2, ..., X_n$ are said to be i.i.d. if they are independent and have the same marginal.
Further,

If Z is a function of several random variables

$$Z = g(X_1, X_2, ..., X_n)$$

the cdf of Z is

$$P[Z \leqslant z] = P[R_z = \{x = (x_1,...,x_n) : g(x) \leqslant z\}]$$

and

$$F_Z(z) = P[X \in R_z] = \int \cdots \int_{x \in R_z} f_{X_1,...,X_n}(x_1^{'},...,x_n^{'})dx_1^{'}...dx_n^{'}$$

The pdf of Z is obtained by taking the derivative of $F_Z(z)$.

24.1.6 Some limits

Definition 24.1.26 *We say that the sequence $F_1, F_2, ...$ of distribution functions converges to the distribution function F, written $F_n \to F$, if $F(x) = \lim_{n\to\infty} F_n(x)$ at each point x where F is continuous.*

We want the sequences $\{F_n\}$ and $\{G_n\}$ to have the same limit in order to drop the requirement that $F_n(x) \to F(x)$ at the point of discontinuity of F.

Theorem 24.1.8 *Continuity*
Suppose that $F_1, F_2, ...$ is a sequence of distribution functions with corresponding characteristic functions $\phi_1, \phi_2, ...$

1. *if $F_n \to F$ for some distribution function F with characteristic function ϕ, then $\phi_n(t) \to \phi(t)$ for all t.*
2. *conversely, if $\phi(t) = \lim_{n\to\phi_n(t)}$ exists and is continuous at $t = 0$, then ϕ is the characteristic function of some distribution function F, and $F_n \to F$.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We can now deal with the convergence of sequence of random variables and discuss the law of large numbers and the central limit theorem.

Definition 24.1.27 *If $X_1, X_2, ..., X$ is a collection of random variables with distributions $F_1, F_2, ..., F$, then we say that X_n converges in distribution to X, written $X_n \xrightarrow{D} X$, if $F_n \to F$.*

We can rewrite this definition in terms of random variables as follow:

Theorem 24.1.9 *Law of large numbers*
Let $X_1, X_2, ...$ be a sequence of i.i.d. random variables with finite means μ. Their partial sums $S_n = \sum_{i=1}^n X_i$ satisfy

$$\frac{1}{n} S_n \xrightarrow{D} \mu \text{ as } n \to \infty$$

We showed that for large n, then S_n is about as big as $n\mu$. So long as the X_i have finite variance, then $S_n - n\mu$ is about as big as $\sqrt{n}$.

Theorem 24.1.10 *Central limit theorem*
Let $X_1, X_2, ...$ be a sequence of i.i.d. random variables with finite means μ and finite non-zero variances σ^2. Their partial sums $S_n = \sum_{i=1}^n X_i$ satisfy

$$\frac{S_n - n\mu}{\sqrt{n\sigma^2}} \xrightarrow{D} N(0,1) \text{ as } n \to \infty$$

That is, the distribution of S_n, suitably normalised to have mean 0 and variance 1, converges to the distribution function of $N(0,1)$. We see that $N(0,1)$ is a distribution and not a random variable. One can show that this result also applies at the level of density functions and mass functions, given some condition of smoothness.

Theorem 24.1.11 *Local limit theorem*
Let $X_1, X_2, ...$ be a sequence of i.i.d. random variables with zero mean and unit variance, and suppose that their common characteristic function ϕ satisfies

$$\int_{-\infty}^{\infty} |\phi(t)|^r dt < \infty$$

for some integer $r \geqslant 1$. The density function g_n of $U_n = n^{-\frac{1}{2}} \sum_{i=1}^n X_i$ exists for $n \geqslant r$, and furthermore

$$g_n(x) \to \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \text{ as } n \to \infty$$

uniformly in $x \in \mathbb{R}$.

24.2 Sampling data

Data sampling is the selection of a subset (a statistical sample) of individuals from within a statistical population to estimate characteristics of the whole population. It is a method allowing researchers to infer information about a population based on results from a subset of the population, without having to investigate every individual. Reducing the number of individuals in a study reduces the cost and workload, and may make it easier to obtain high quality information, but this has to be balanced against having a large enough sample size with enough power to detect a true association. The two main advantages of sampling are lower cost and faster data collection than measuring the entire population.
Each observation measures one or more properties (such as weight, location, colour) of observable bodies distinguished as independent objects or individuals. In survey sampling, weights can be applied to the data to adjust for the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

sample design, particularly stratified sampling. Results from probability theory and statistical theory are employed to guide the practice.
As a remedy, we seek a sampling frame which has the property that we can identify every single element and include any in our sample. The most straightforward type of frame is a list of elements of the population (preferably the entire population) with appropriate contact information.

If a sample is to be used, by whatever method it is chosen, it is important that the individuals selected are representative of the whole population. This may involve specifically targeting hard to reach groups.

There are several different sampling techniques available, and they can be subdivided into two groups:

1. Probability sampling (randomisation): In probability (random) sampling, you start with a complete sampling frame of all eligible individuals from which you select your sample. In this way, all eligible individuals have a chance of being chosen for the sample, and you will be more able to generalise the results from your study. Probability sampling methods tend to be more time-consuming and expensive than non-probability sampling.

2. Non-probability sampling: you do not start with a complete sampling frame, so some individuals have no chance of being selected. Consequently, you cannot estimate the effect of sampling error and there is a significant risk of ending up with a non-representative sample which produces non-generalisable results. However, non-probability sampling methods tend to be cheaper and more convenient, and they are useful for exploratory research and hypothesis generation.

Probability sampling includes: Simple Random Sampling, Systematic Sampling, Stratified Sampling, Probability Proportional to Size Sampling, and Cluster or Multistage Sampling. These various ways of probability sampling have two things in common:

1. Every element has a known nonzero probability of being sampled

2. Every element involves random selection at some point.

Within any of the types of frames, a variety of sampling methods can be employed, individually or in combination. Factors commonly influencing the choice between these designs include:

- Nature and quality of the frame
- Availability of auxiliary information about units on the frame
- Accuracy requirements, and the need to measure accuracy
- Whether detailed analysis of the sample is expected
- Cost/operational concerns

24.2.1 Probability sampling

24.2.1.1 Simple random sampling

In a simple random sample (SRS) of a given size, all such subsets of the frame are given an equal probability. Thus, each element of the frame has an equal probability of selection: the frame is not subdivided partitioned. SRS refers to any sampling method that has the following properties:

- The population consists of N objects.
- The sample consists of n objects.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- If all possible samples of n objects are equally likely to occur, the sampling method is called simple random sampling. See Figure (24.1).

Furthermore, any given pair of elements has the same chance of selection as any other such pair (and similarly for triples, and so on). This minimises bias and simplifies analysis of results. In particular, the variance between individual results within the sample is a good indicator of variance in the overall population, which makes it relatively easy to estimate the accuracy of results.

Figure 24.1: Selecting a simple random sample

As with all probability sampling methods, SRS allows the sampling error to be calculated and reduces selection bias. A specific advantage is that it is the most straightforward method of probability sampling. On the other hand we may not select enough individuals with our characteristic of interest, especially if that characteristic is uncommon. Further, SRS can be vulnerable to sampling error because the randomness of the selection may result in a sample that does not reflect the makeup of the population. For instance, a simple random sample of ten people from a given country will on average produce five men and five women, but any given trial is likely to overrepresent one sex and underrepresent the other. Systematic and stratified techniques attempt to overcome this problem by using information about the population to choose a more representative sample.

SRS may also be cumbersome and tedious when sampling from an unusually large target population. In some cases, investigators are interested in research questions specific to subgroups of the population. For example, researchers might be interested in examining whether cognitive ability as a predictor of job performance is equally applicable across racial groups. SRS cannot accommodate the needs of researchers in this situation because it does not provide subsamples of the population. Stratified sampling addresses this weakness of SRS.

24.2.1.2 Systematic sampling

Systematic sampling (or interval sampling) relies on arranging the study population according to some ordering scheme and then selecting elements at regular intervals through that ordered list. See Figure (24.2). It involves a random start and then proceeds with the selection of every kth element from then onwards. In this case, $k = \frac{\text{population size}}{\text{sample size}}$. It is important that the starting point is not automatically the first in the list, but is instead randomly chosen from within the first to the kth element in the list.
As long as the starting point is randomised, systematic sampling is a type of probability sampling. It is easy to imple-

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

ment and the stratification induced can make it efficient, if the variable by which the list is ordered is correlated with the variable of interest.

Figure 24.2: Selecting a random sample using systematic sampling

Systematic sampling is often more convenient than simple random sampling, and it is easy to administer. However, it is especially vulnerable to periodicities in the list. If periodicity is present and the period is a multiple or factor of the interval used, the sample is especially likely to be unrepresentative of the overall population, making the scheme less accurate than simple random sampling. For example, if a group of students were being sampled to gain their opinions on college facilities, but the Student Record Department's central list of all students was arranged such that the sex of students alternated between male and female, choosing an even interval (e.g. every 20th student) would result in a sample of all males or all females. Whilst in this example the bias is obvious and should be easily corrected, this may not always be the case.
Another drawback of systematic sampling is that even in scenarios where it is more accurate than SRS, its theoretical properties make it difficult to quantify that accuracy. As described above, systematic sampling is an EPS method, because all elements have the same probability of selection

24.2.1.3 Stratified sampling

When the population embraces a number of distinct categories, the frame can be organised by these categories into separate strata. Each stratum is then sampled as an independent sub-population, out of which individual elements can be randomly selected. The ratio of the size of this random selection (or sample) to the size of the population is called a sampling fraction. See Figure (24.3).
It is used when we might reasonably expect the measurement of interest to vary between the different subgroups, and we want to ensure representation from all the subgroups. For example, in a study of stroke outcomes, we may stratify the population by sex, to ensure equal representation of men and women. The study sample is then obtained by taking equal sample sizes from each stratum. In stratified sampling, it may also be appropriate to choose non-equal sample sizes from each stratum. For example, in a study of the health outcomes of nursing staff in a county, if there are three hospitals each with different numbers of nursing staff (hospital A has 500 nurses, hospital B has 1000 and hospital C has 2000), then it would be appropriate to choose the sample numbers from each hospital proportionally (for example, 10 from hospital A, 20 from hospital B and 40 from hospital C). This ensures a more realistic and accurate estimation of the health outcomes of nurses across the county, whereas simple random sampling would over-represent nurses from hospitals A and B. The fact that the sample was stratified should be taken into account at the analysis stage.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 24.3: Selecting a random sample using stratified sampling

There are several potential benefits to stratified sampling such as:

1. dividing the population into distinct, independent strata can enable researchers to draw inferences about specific subgroups that may be lost in a more generalized random sample.
2. utilising a stratified sampling method can lead to more efficient statistical estimates (provided that strata are selected based upon relevance to the criterion in question, instead of availability of the samples). Even if a stratified sampling approach does not lead to increased statistical efficiency, such a tactic will not result in less efficiency than would simple random sampling, provided that each stratum is proportional to the group's size in the population.
3. it is sometimes the case that data are more readily available for individual, pre-existing strata within a population than for the overall population; in such cases, using a stratified sampling approach may be more convenient than aggregating data across groups (though this may potentially be at odds with the previously noted importance of utilizing criterion-relevant strata).
4. since each stratum is treated as an independent population, different sampling approaches can be applied to different strata, potentially enabling researchers to use the approach best suited (or most cost-effective) for each identified subgroup within the population.

There are, however, some potential drawbacks to using stratified sampling.

1. identifying strata and implementing such an approach can increase the cost and complexity of sample selection, as well as leading to increased complexity of population estimates.
2. when examining multiple criteria, stratifying variables may be related to some, but not to others, further complicating the design, and potentially reducing the utility of the strata.
3. in some cases (such as designs with a large number of strata, or those with a specified minimum sample size per group), stratified sampling can potentially require a larger sample than would other methods (although in most cases, the required sample size would be no larger than would be required for simple random sampling).

A stratified sampling approach is most effective when three conditions are met

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

1. Variability within strata are minimised
2. Variability between strata are maximised
3. The variables upon which the population is stratified are strongly correlated with the desired dependent variable.

Some advantages over other sampling methods

1. Focuses on important subpopulations and ignores irrelevant ones.
2. Allows use of different sampling techniques for different subpopulations.
3. Improves the accuracy/efficiency of estimation.
4. Permits greater balancing of statistical power of tests of differences between strata by sampling equal numbers from strata varying widely in size.

Some disadvantages are

1. It requires selection of relevant stratification variables which can be difficult.
2. It is not useful when there are no homogeneous subgroups.
3. It can be expensive to implement.

24.2.1.4 Cluster sampling

Sometimes it is more cost-effective to select respondents in groups (or clusters). In a clustered sample, subgroups of the population are used as the sampling unit, rather than individuals. The population is divided into subgroups, which are randomly selected to be included in the study. See Figure (24.4). Sampling is often clustered by geography, or by time periods. For instance, if surveying households within a city, we might choose to select 100 city blocks and then interview every household within the selected blocks.
Clustering can reduce travel and administrative costs. In the example above, an interviewer can make a single trip to visit several households in one block, rather than having to drive to a different block for each household.

Figure 24.4: Selecting a random sample using cluster sampling

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

It also means that one does not need a sampling frame listing all elements in the target population. Instead, clusters can be chosen from a cluster-level frame, with an element-level frame created only for the selected clusters. In the example above, the sample only requires a block-level city map for initial selections, and then a household-level map of the 100 selected blocks, rather than a household-level map of the whole city.

Cluster sampling generally increases the variability of sample estimates above that of simple random sampling, depending on how the clusters differ between one another as compared to the within-cluster variation. For this reason, cluster sampling requires a larger sample than SRS to achieve the same level of accuracy, but cost savings from clustering might still make this a cheaper option.

Cluster sampling is commonly implemented as multistage sampling. This is a complex form of cluster sampling in which two or more levels of units are embedded one in the other.

1. The first stage consists of constructing the clusters that will be used to sample from.
2. In the second stage, a sample of primary units is randomly selected from each cluster (rather than using all units contained in all selected clusters).
3. In following stages, in each of those selected clusters, additional samples of units are selected, and so on.
4. All ultimate units (individuals, for instance) selected at the last step of this procedure are then surveyed.

Thus, this technique is essentially the process of taking random subsamples of preceding random samples. See Figure (24.5).

Figure 24.5: Selecting a random sample using cluster sampling

Multistage sampling can substantially reduce sampling costs, where the complete population list would need to be constructed (before other sampling methods could be applied). By eliminating the work involved in describing clusters that are not selected, multistage sampling can reduce the large costs associated with traditional cluster sampling. However, each sample may not be a full representative of the whole population.

24.2.2 Non-probability sampling

24.2.2.1 Convenience sampling

Convenience sampling is perhaps the easiest method of sampling, because participants are selected based on availability and willingness to take part. Useful results can be obtained, but the results are prone to significant bias, because

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

those who volunteer to take part may be different from those who choose not to (volunteer bias), and the sample may not be representative of other characteristics, such as age or sex. Note: volunteer bias is a risk of all non-probability sampling methods.

24.2.2.2 Quota sampling

In quota sampling, the population is first segmented into mutually exclusive sub-groups, just as in stratified sampling. Then judgement is used to select the subjects or units from each segment based on a specified proportion. For example, an interviewer may be told to sample 200 females and 300 males between the age of 45 and 60.
It is this second step which makes the technique one of non-probability sampling. In quota sampling the selection of the sample is non-random. For example, interviewers might be tempted to interview those who look most helpful. The problem is that these samples may be biased because not everyone gets a chance of selection. This random element is its greatest weakness and quota versus probability has been a matter of controversy for several years.

24.2.2.3 Judgement sampling

Also known as selective, or subjective, sampling, this technique relies on the judgement of the researcher when choosing who to ask to participate. Researchers may implicitly thus choose a representative sample to suit their needs, or specifically approach individuals with certain characteristics. This approach is often used by the media when canvassing the public for opinions and in qualitative research.
Judgement sampling has the advantage of being time-and cost-effective to perform whilst resulting in a range of responses (particularly useful in qualitative research). However, in addition to volunteer bias, it is also prone to errors of judgement by the researcher and the findings, whilst being potentially broad, will not necessarily be representative.

24.2.2.4 Minimax sampling

In imbalanced datasets, where the sampling ratio does not follow the population statistics, one can resample the dataset in a conservative manner called minimax sampling. The minimax sampling has its origin in Anderson minimax ratio whose value is proved to be 0.5: in a binary classification, the class-sample sizes should be chosen equally. This ratio can be proved to be minimax ratio only under the assumption of LDA classifier with Gaussian distributions. The notion of minimax sampling is recently developed for a general class of classification rules, called class-wise smart classifiers.

24.2.2.5 Accidental sampling

Accidental sampling (also known as grab, convenience or opportunity sampling) is a type of nonprobability sampling which involves the sample being drawn from that part of the population which is close to hand. That is, a population is selected because it is readily available and convenient. It may be through meeting the person or including a person in the sample when one meets them or chosen by finding them through technological means such as the internet or through phone. The researcher using such a sample cannot scientifically make generalizations about the total population from this sample because it would not be representative enough. For example, if the interviewer were to conduct such a survey at a shopping center early in the morning on a given day, the people that he/she could interview would be limited to those given there at that given time, which would not represent the views of other members of society in such an area, if the survey were to be conducted at different times of day and several times per week. This type of sampling is most useful for pilot testing. Several important considerations for researchers using convenience samples include:

- Are there controls within the research design or experiment which can serve to lessen the impact of a non-random convenience sample, thereby ensuring the results will be more representative of the population?
- Is there good reason to believe that a particular convenience sample would or should respond or behave differently than a random sample from the same population?

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Is the question being asked by the research one that can adequately be answered using a convenience sample?

In social science research, snowball sampling is a similar technique, where existing study subjects are used to recruit more subjects into the sample. Some variants of snowball sampling, such as respondent driven sampling, allow calculation of selection probabilities and are probability sampling methods under certain conditions.

24.2.2.6 Voluntary sampling

The voluntary sampling method is a type of non-probability sampling. Volunteers choose to complete a survey. They may be invited through advertisements in social media. The target population for advertisements can be selected by characteristics like location, age, sex, income, occupation, education or interests using tools provided by the social medium. The advertisement may include a message about the research and link to a survey. After following the link and completing the survey the volunteer submits the data to be included in the sample population. This method can reach a global population but is limited by the campaign budget. Volunteers outside the invited population may also be included in the sample.
It is difficult to make generalisations from this sample because it may not represent the total population. Often, volunteers have a strong interest in the main topic of the survey.

24.2.2.7 Panel sampling

Panel sampling is the method of first selecting a group of participants through a random sampling method and then asking that group for (potentially the same) information several times over a period of time. Therefore, each participant is interviewed at two or more time points; each period of data collection is called a wave. The method was developed by sociologist Paul Lazarsfeld in 1938 as a means of studying political campaigns. This longitudinal sampling-method allows estimates of changes in the population, for example with regard to chronic illness to job stress to weekly food expenditures. Panel sampling can also be used to inform researchers about within-person health changes due to age or to help explain changes in continuous dependent variables such as spousal interaction. There have been several proposed methods of analysing panel data, including MANOVA, growth curves, and structural equation modeling with lagged effects.

24.3 Some random sampling

For details see text book by Rice [1995]. We assume that the population is of size N and that associated with each member of the population is a numerical value of interest denoted by $x_1, x_2, .., x_N$. The variable x_i may be a numerical variable such as age or weight, or it may take on the value 1 or 0 to denote the presence or absence of some characteristic. The latter is called the dichotomous case.

24.3.1 The sample moments

In general, the population variance of a finite population of size N is given by

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \text{ with } \mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

where μ is the population mean. In the dichotomous case μ equals the proportion p of individuals in the population having the particular characteristic. The population total is

$$\tau = \sum_{i=1}^{N} x_i = N\mu$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

In many practical situations, the true variance of a population is not known a priori and must be computed somehow. When dealing with extremely large populations, it is not possible to count every object in the population and one must estimate the variance of a population from a sample.
We take a sample with replacement (SWR) of n values $X_1, ..., X_n$ from the population, where $n < N$ and such that X_i is a random variable (representative of the population). X_i is the value of the ith member of the sample, and x_i is that of the ith member of the population. The joint distribution of the X_i is determined by that of the x_i. We let $\xi_1, .., \xi_m$ be the elements of a vector corresponding to the possible values of x_i. Since each member of the population is equally likely to be in the sample, we get

$$P(X_i = \xi_j) = \frac{n_j}{N}$$

The sample mean is

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

Theorem 24.3.1 *With simple random sampling* $E[\overline{X}] = \mu$

We say that an estimate is unbiased if its expectation equals the quantity we wish to estimate.

$$\text{Mean squared error } = \text{ variance } + (biased)^2$$

Since $\overline{X}$ is unbiased, its mean square error is equal to its variance.

Lemma 24.3.1 *With simple random sampling*

$$Cov(X_i, X_j) = \begin{cases} \sigma^2 \; if \; i = j \\ -\frac{\sigma^2}{(N-1)} \; if \; i \neq j \end{cases}$$

It shows that X_i and X_j are not independent of each other for $i \neq j$, but that the covariance is very small for large values of N.

Theorem 24.3.2 *With simple random sampling*

$$Var(\overline{X}) = \frac{\sigma^2}{n}\big(1 - \frac{n-1}{N-1}\big)$$

Note, the ratio $\frac{n}{N}$ is called the sampling fraction and

$$p_c = \big(1 - \frac{n-1}{N-1}\big)$$

is the finite population correction. If the sampling fraction is very small we get the approximation

$$\sigma_{\overline{X}} \approx \frac{\sigma}{\sqrt{n}}$$

We estimate the variance on the basis of this sample as

$$\hat{\sigma}^2 = \sigma_n^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2 \text{ with } \overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

This is the biased sample variance. The unbiased sample variance is

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\overline{\sigma}_n^2 = \frac{1}{n-1}\sum_{i=1}^n (X_i - \overline{X})^2 \text{ with } \overline{X} = \frac{1}{n}\sum_{i=1}^n X_i$$

While the first one may be seen as the variance of the sample considered as a population (over n), the second one is the unbiased estimator of the population variance (over N) where $(n-1)$ is the Bessel's correction. Put another way, we get

$$E[\sigma_n^2] = \frac{n-1}{n}\frac{N}{N-1}\sigma^2 \text{ and } E[\overline{\sigma}_n^2] = \sigma^2$$

That is, the unbiased estimate of σ^2 may be obtained by multiplying σ_n^2 by the factor $\frac{n}{n-1}\frac{N-1}{N}$. If the population is large relative to n, the dominant bias is due to the term $\frac{n-1}{n}$. In general one set n between 50 and 180 days. Being a function of random variables, the sample variance is itself a random variable, and it is natural to study its distribution. In the case that y_i are independent observations from a normal distribution, Cochran's theorem shows that $\overline{\sigma}_n^2$ follows a scaled chi-squared distribution

$$(n-1)\frac{\overline{\sigma}_n^2}{\sigma^2} \approx \chi_{n-1}^2$$

If the conditions of the law of large numbers hold for the squared observations, $\overline{\sigma}_n^2$ is a consistent estimator of σ^2 and the variance of the estimator tends asymptotically to zero. The obtained standard deviation $\tilde{\sigma}_N$ is an estimator of σ called the historical volatility.

Corollary 5 *An unbiased estimate of $Var(\overline{X})$ is*

$$s_{\overline{X}}^2 = \frac{\hat{\sigma}^2}{n}\frac{n}{n-1}\frac{N-1}{N}\frac{N-n}{N-1} = \frac{s^2}{n}(1-\frac{n}{N})$$

where

$$s^2 = \frac{1}{n-1}\sum_{i=1}^n (X_i - \overline{X})^2$$

One can compute the sample mean and sample variance recursively as follow. Starting from the identities

$$(n+1)\overline{X}_{n+1} = n\overline{X}_n + X_{n+1}$$

and

$$(n+1)\big(\sigma_{n+1}^2 + (\overline{X}_{n+1})^2\big) = n\big(\sigma_n^2 + (\overline{X}_n)^2\big) + X_{n+1}^2$$

we get the recursive sample mean

$$\overline{X}_{n+1} = \overline{X}_n + \frac{X_{n+1} - \overline{X}_n}{n+1} \tag{24.3.17}$$

and the recursive sample variance

$$\sigma_{n+1}^2 = \sigma_n^2 + (\overline{X}_n)^2 - (\overline{X}_{n+1})^2 + \frac{X_{n+1}^2 - \sigma_n^2 - (\overline{X}_n)^2}{n+1} \tag{24.3.18}$$

such that σ_{n+1}^2 only depends on σ_n^2, $\overline{X}_n$, $\overline{X}_{n+1}$ and X_{n+1}. If we normalise the data with a constant K such that the ith value is $\frac{X_i}{K}$, we let $\tilde{X} = \frac{1}{n}\sum_{i=1}^n \frac{X_i}{K}$ be the sample mean and $\tilde{\sigma}_n^2 = \frac{1}{n}\sum_{i=1}^n (\frac{X_i}{K} - \tilde{X})^2$ be the sample variance. Then, the recursive sample mean becomes

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\tilde{X}_{n+1} = \tilde{X}_n + \frac{1}{n+1}\left(\frac{X_{n+1}}{K} - \tilde{X}_n\right) \tag{24.3.19}$$

and the recursive sample variance becomes

$$\tilde{\sigma}^2_{n+1} = \tilde{\sigma}^2_n + (\tilde{X}_n)^2 - (\tilde{X}_{n+1})^2 + \frac{1}{n+1}\left(\frac{X^2_{n+1}}{K^2} - \tilde{\sigma}^2_n - (\tilde{X}_n)^2\right) \tag{24.3.20}$$

24.3.2 Estimation of a ratio

We consider the estimation of a ratio, and assume that for each member of a population, two values, x and y, may be measured. The ratio of interest is

$$r = \frac{\sum_{i=1}^N y_i}{\sum_{i=1}^N x_i} = \frac{\mu_y}{\mu_x}$$

Assuming that a sample is drawn consisting of the pairs (X_i, Y_i), the natural estimate of r is $R = \frac{\overline{Y}}{\overline{X}}$. Since R is a nonlinear function of the random variables $\overline{X}$ and $\overline{Y}$, there is no closed form for $E[R]$ and $Var(R)$ and we must approximate them by using $Var(\overline{X})$, $Var(\overline{Y})$, and $Cov(\overline{X}, \overline{Y})$. We define the the population covariance of x and y as

$$\sigma_{xy} = \frac{1}{N}\sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y)$$

One can show that

$$Cov(\overline{X}, \overline{Y}) = \frac{\sigma_{xy}}{n} p_c$$

Theorem 24.3.3 *With simple random sampling, the approximate variance of $R = \frac{\overline{Y}}{\overline{X}}$ is*

$$Var(R) \approx \frac{1}{\mu_x^2}(r^2\sigma^2_{\overline{X}} + \sigma^2_{\overline{Y}} - 2r\sigma_{\overline{XY}}) = \frac{1}{n}p_c\frac{1}{\mu_x^2}(r^2\sigma_x^2 + \sigma_y^2 - 2r\sigma_{xy})$$

The population correlation coefficient given by

$$\rho = \frac{\sigma_{xy}}{\sigma_x\sigma_y}$$

is a measure of the strength of the linear relationship between the x and the y values in the population. We can express the variance in the above theorem in terms of ρ as

$$Var(R) \approx \frac{1}{n}p_c\frac{1}{\mu_x^2}(r^2\sigma_x^2 + \sigma_y^2 - 2r\rho\sigma_x\sigma_y)$$

so that strong correlation of the same sign as r decreases the variance. Further, for small μ_x we get large variance, since small values of $\overline{X}$ in the ratio $R = \frac{\overline{Y}}{\overline{X}}$ cause R to fluctuate wildly.

Theorem 24.3.4 *With simple random sampling, the expectation of R is given approximately by*

$$E[R] \approx r + \frac{1}{n}p_c\frac{1}{\mu_x^2}(r^2\sigma_x^2 - \rho\sigma_x\sigma_y)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

so that strong correlation of the same sign as r decreases the bias, and the bias is large if μ_x is small. In addition, the bias is of the order $\frac{1}{n}$, so its contribution to the mean squared error is of the order $\frac{1}{n^2}$ while the contribution of the variance is of order $\frac{1}{n}$. Hence, for large sample, the bias is negligible compared to the standard error of the estimate. For large samples, truncating the Taylor Series after the linear term provides a good approximation, since the deviations $\overline{X} - \mu_x$ and $\overline{Y} - \mu_y$ are likely to be small. To this order of approximation, R is expressed as a linear combination of $\overline{X}$ and $\overline{Y}$, and an argument based on the central limit theorem can be used to show that R is approximately normally distributed, and confidence intervals can be formed for r by using the normal distribution. In order to estimate the standard error of R, we substitute R for r in the formula of the above theorem where the x and y population variances are estimated by s_x^2 and s_y^2. The population covariance is estimated by

$$s_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y}) = \frac{1}{n-1}\Big(\sum_{i=1}^{n} X_iY_i - n\overline{X}\overline{Y}\Big)$$

and the population correlation is estimated by

$$\widehat{\rho} = \frac{s_{xy}}{s_x s_y}$$

The estimated variance of R is thus

$$s_R^2 = \frac{1}{n}p_c\frac{1}{\overline{X}^2}(R^2 s_x^2 + s_y^2 - 2Rs_{xy})$$

and the approximate $100(1-\alpha)\%$ confidence interval for r is $R \pm z(\frac{\alpha}{2})s_R$.

24.3.3 Stratified random sampling

The population is partitioned into subpopulations, or strata, which are then independently sampled. We are interested in obtaining information about each of a number of natural subpopulations in addition to information about the population as a whole. It guarantees a prescribed number of observations from each subpopulation, whereas the use of a simple random sample can result in underrepresentation of some subpopulations. Further, the stratified sample mean can be considerably more precise than the mean of a simple random sample, especially if there is considerable variation between strata.

We will denote by N_l, where $l = 1, .., L$ the population sizes in the L strata such that $N_1 + N_2 + .. + N_L = N$ the total population size. The population mean and variance of the lth stratum are denoted by μ_l and σ_l^2 (unknown). The overall population mean can be expressed in terms of the μ_l as follows

$$\mu = \frac{1}{N}\sum_{l=1}^{L}\sum_{i=1}^{N_l} x_{il} = \frac{1}{N}\sum_{l=1}^{L} N_l\mu_l = \sum_{l=1}^{L} W_l\mu_l$$

where x_{il} denotes the ith population value in the lth stratum and $W_l = \frac{N_l}{N}$ is the fraction of the population contained in the lth stratum.

Within each stratum a simple random sample of size n_l is taken. The sample mean in stratum l is denoted by

$$\overline{X}_l = \frac{1}{n_l}\sum_{i=1}^{n_l} X_{il}$$

where X_{il} denotes the ith observation in the lth stratum. By analogy with the previous calculation we get

$$\overline{X}_s = \sum_{l=1}^{L}\frac{N_l\overline{X}_l}{N} = \sum_{l=1}^{L} W_l\overline{X}_l$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 24.3.5 *The stratisfied estimate, $\overline{X}_s$ of the population mean is unbiased. That is, $E[\overline{X}_s] = \mu$.*

Since we assume that the samples from different strata are independent of one another and that within each stratum a simple random sample is taken, the variance of $\overline{X}_s$ can easily be calculate.

Theorem 24.3.6 *The variance of the stratisfied sample mean is given by*

$$Var(\overline{X}_s) = \sum_{l=1}^{L} W_l^2 \frac{1}{n_l}(1 - \frac{n_l - 1}{N_l - 1})\sigma_l^2 \tag{24.3.21}$$

Note, $\frac{n_l-1}{N_l-1}$ represents the finite population corrections. If the sampling fractions within all strata are small ($\frac{n_l-1}{N_l-1} << 1$), we get the approximation

$$Var(\overline{X}_s) \approx \sum_{l=1}^{L} \frac{W_l^2}{n_l}\sigma_l^2 \tag{24.3.22}$$

The estimate of σ_l^2 is given by

$$S_l^2 = \frac{1}{n_l - 1}\sum_{i=1}^{n_l}(X_{il} - \overline{X}_l)^2$$

and $Var(\overline{X}_s)$ is estimated by

$$S^2_{\overline{X}_s} = \sum_{l=1}^{L} W_l^2 \frac{1}{n_l}(1 - \frac{n_l - 1}{N_l - 1})S_l^2$$

The question that naturally arises is how to choose $n_1, .., n_L$ to minimise $Var(\overline{X}_s)$ subject to the constraint $n_1 + .. + n_L = n$. Ignoring the finite population correction within each stratum we get the Neyman allocation.

Theorem 24.3.7 *The sample sizes $n_1, .., n_L$ that minimise $Var(\overline{X}_s)$ subject to the constraint $n_1 + .. + n_L = n$ are given by*

$$n_l = n\frac{W_l\sigma_l}{\sum_{k=1}^{L} W_k\sigma_k}$$

where $l = 1, .., L$.

This theorem shows that those strata for which $W_l\sigma_l$ is large should be sampled heavily. If W_l is large, the stratum contains a large fraction of the population. If σ_l is large, the population values in the stratum are quite variable, and a relatively large sample size must be used. Substituting the optimal values of n_l into the Equation (24.3.21) we get the following corollary.

Corollary 6 *Denoting by $\overline{X}_{so}$ the stratified estimate using the optimal Neyman allocation and neglecting the finite population correction, we get*

$$Var(\overline{X}_{so}) = \frac{1}{n}\left(\sum_{l=1}^{L} W_l\sigma_l\right)^2$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The optimal allocations depend on the individual variances of the strata, which generally will not be known. A simple alternative method of allocation is to use the same sampling fraction in each stratum

$$\frac{n_1}{N_1} = \frac{n_2}{N_2} = ... = \frac{n_L}{N_L}$$

which holds if

$$n_l = n\frac{N_L}{N} = nW_l \tag{24.3.23}$$

$l = 1, .., L$. This method is called the Proportional allocation. The estimate of the population mean based on proportional allocation is

$$\overline{X}_{sp} = \sum_{l=1}^{L} W_l \overline{X}_l = \frac{1}{n}\sum_{l=1}^{L}\sum_{i=1}^{n_l} X_{il}$$

since $\frac{W_l}{n_l} = \frac{1}{n}$. This estimate is simply the unweighted mean of the sample values.

Theorem 24.3.8 *With stratified sampling based on proportional allocation, ignoring the finite population correction, we get*

$$Var(\overline{X}_{sp}) = \frac{1}{n}\sum_{l=1}^{L} W_l \sigma_l^2$$

We can compare $Var(\overline{X}_{so})$ and $Var(\overline{X}_{sp})$ to define when optimal allocation is substantially better than proportional allocation.

Theorem 24.3.9 *With stratified random sampling, the difference between the variance of the estimate of the population mean based on proportional allocation and the variance of that estimate based on optimal allocation is, ignoring the finite population correction,*

$$Var(\overline{X}_{sp}) - Var(\overline{X}_{so}) = \frac{1}{n}\sum_{l=1}^{L} W_l(\sigma_l - \overline{\sigma})^2$$

where

$$\overline{\sigma} = \sum_{l=1}^{L} W_l \sigma_l$$

As a result, if the variances of the strata are all the same, proportional allocation yields the same results as optimal allocation. The more variable these variances are, the better it is to use optimal allocation.

We can also compare the variance under simple random sampling with the variance under proportional allocation. Neglecting the finite population correction, the variance under simple random sampling is $Var(\overline{X}) = \frac{\sigma^2}{n}$. We first need a relationship between the overall population variance σ^2 and the strata variances σ_l^2. The overall population variance may be expressed as

$$\sigma^2 = \frac{1}{N}\sum_{l=1}^{L}\sum_{i=1}^{N_L}(x_{il} - \mu)^2$$

Further, using the relation

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$(x_{il} - \mu)^2 = (x_{il} - \mu_l)^2 + 2(x_{il} - \mu_l)(\mu_l - \mu) + (\mu_l - \mu)^2$$

and realising that when both sides are summed over l, the middle term on the right-hand side becomes zero, we have

$$\sum_{i=1}^{N_L}(x_{il} - \mu)^2 = \sum_{i=1}^{N_L}(x_{il} - \mu_l)^2 + N_L(\mu_l - \mu)^2 = N_L\sigma_l^2 + N_L(\mu_l - \mu)^2$$

Dividing both sides by N and summing over l, we have

$$\sigma^2 = \sum_{l=1}^{L} W_l\sigma_l^2 + \sum_{l=1}^{L} W_l(\mu_l - \mu)^2$$

Substituting this expression for σ^2 into $Var(\overline{X})$ and using the formula for $Var(\overline{X}_{sp})$ completes the proof of the following theorem.

Theorem 24.3.10 *The difference between the variance of the mean of a simple random sample and the variance of the mean of a stratified random sample based on proportional allocation is, neglecting the finite population correction,*

$$Var(\overline{X}) - Var(\overline{X}_{sp}) = \frac{1}{n}\sum_{l=1}^{L} W_l(\mu_l - \mu)^2$$

Thus, stratified random sampling with proportional allocation always gives a smaller variance than does simple random sampling, providing that the finite population correction is ignored. Typically, stratified random sampling can result in substantial increases in precision for populations containing values that vary greatly in size.

In order to construct the optimal number of strata, the population values themselves (which are unknown) would have to be used. Stratification must therefore be done on the basis of some related variable that is known or on the results of earlier samples.

According to the Neyman-Pearson Paradigm, a decision as to weather or not to reject H_0 in favour of H_A is made on the basis of $T(X)$, where X denotes the sample values and $T(X)$ is a statistic.

The statistical properties of the methods are relevant if it is reasonable to model the data stochastically. There exists methods that are sample analogues of the cumulative distribution function of a random variable. It is useful in displaying the distribution of sample values.
Given $x_1, .., x_n$ a batch of numbers, the empirical cumulative distribution function (ecdf) is defined as

$$F_n(x) = \frac{1}{n}(nox_i \leqslant x)$$

where $F_n(x)$ gives the proportion of the data less than or equal to x. It is a step function with a jump of height $\frac{1}{n}$ at each point x_i. The ecdf is to a sample what the cumulative distribution is to a random variable. We now consider some of the elementary statistical properties of the ecdf when $X_1, .., X_n$ is a random sample from a continuous distribution function F. We choose to express F_n as

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{(-\infty,x]}(X_i)$$

The random variables $I_{(-\infty,x]}(X_i)$ are independent Bernoulli random variables

$$I_{(-\infty,x]}(X_i) = \begin{cases} 1 \text{ with probability } F(x) \\ 0 \text{ with probability } 1 - F(x) \end{cases}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Thus, $nF_n(x)$ is a binomial random variable with the first two moments being

$$E[F_n(x)] = F(x)$$
$$Var(F_n(x)) = \frac{1}{n}F(x)[1 - F(x)]$$

24.3.4 Geometric mean

The use of a geometric mean normalises the ranges being averaged, so that no range dominates the weighting, and a given percentage change in any of the properties has the same effect on the geometric mean. The geometric mean applies only to positive numbers. It is also often used for a set of numbers whose values are meant to be multiplied together or are exponential in nature, such as data on the growth of the human population or interest rates of a financial investment. The geometric mean of a data set $\{a_1, a_2, .., a_n\}$ is given by

$$(\prod_{i=1}^{n} a_i)^{\frac{1}{n}}$$

The geometric mean of a data set is less than the data set's arithmetic mean unless all members of the data set are equal, in which case the geometric and arithmetic means are equal. By using logarithmic identities to transform the formula, the multiplications can be expressed as a sum and the power as a multiplication

$$(\prod_{i=1}^{n} a_i)^{\frac{1}{n}} = e^{\frac{1}{n}\sum_{i=1}^{n} \ln a_i}$$

This is sometimes called the log-average. It is simply computing the arithmetic mean of the logarithm-transformed values of a_i (i.e., the arithmetic mean on the log scale) and then using the exponentiation to return the computation to the original scale.

24.4 Some continuous variables and their distributions

For details see text book by Grimmett et al. [1992].

24.4.1 Some popular distributions

24.4.1.1 Uniform distribution

The continuous uniform distribution is a family of symmetric probability distributions such that for each member of the family, all intervals of the same length on the distribution's support are equally probable. The support is defined by two parameters a and b, which are its minimum and maximum values. The distribution $U(a, b)$ is the maximum entropy probability distribution for a random variate X under the constraint that it is in the distribution's support. X is uniform on $[a, b]$ with $-\infty < a < b < \infty$ if

$$F(x) = \begin{cases} 0 \text{ if } x \leqslant a \\ \frac{(x-a)}{(b-a)} \text{ if } a < x \leqslant b \\ 1 \text{ if } x > b \end{cases}$$

Roughly speaking, X takes any value between a and b with equal probability. The probability density function is given by

$$f(x) = \begin{cases} \frac{1}{(b-a)} \text{ for } x \in [a, b] \\ 0 \text{ otherwise} \end{cases}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The first moment of the distribution is $E[X] = \frac{1}{2}(a+b)$ and the second centralised moment is $Var(X) = \frac{1}{12}(b-a)^2$.

24.4.1.2 Exponential distribution

The exponential distribution is the probability distribution describing the time between events in a Poisson process, that is, a process where events occur continuously and independently at a constant average rate. It is the continuous analogue of the geometric distribution, and it has the key property of being memoryless. X is exponential with parameter $\lambda(>0)$ if

$$F(x) = \begin{cases} 1 - e^{-\lambda x}, x \geqslant 0 \\ 0 \text{ otherwise} \end{cases}$$

which is the continuous limit of the waiting time distribution. The probability density function is given by

$$f(x) = \begin{cases} \lambda e^{-\lambda x} \text{ for } x \geqslant 0 \\ 0 \text{ otherwise} \end{cases}$$

The mean of X is given by

$$E[X] = \int_0^\infty [1 - F(x)]dx = \frac{1}{\lambda}$$

and the variance of X is given by $Var(X) = \frac{1}{\lambda^2}$ so that the standard deviation of X is equal to its mean. The moments of X for $n = 1, 2, ..$ are given by

$$E[X^n] = \frac{n!}{\lambda^n}$$

An exponentially distributed random variable T follows the relation

$$P(T > s + t | T > s) = P(T > t), \forall s, t \geqslant 0$$

24.4.1.3 Gamma distribution

The gamma distribution is a two-parameter family of continuous probability distributions where the exponential distribution and the chi-squared distribution are special cases. It is the maximum entropy probability distribution for a random variable X for which $E[X] = k\theta = \frac{\alpha}{\beta}$ is fixed and greater than zero, and $E[\ln X] = \psi(k) + \ln\theta = \psi(\alpha) - \ln\beta$ is fixed and $\psi(\cdot)$ is the digamma function. In the following we denote the pair (k, θ) by $(t, \frac{1}{\lambda})$ corresponding to $\alpha = t$ and $\beta = \lambda$. The variable X has the gamma distribution with parameters λ, $t > 0$, denoted by $\Gamma(\lambda, t)$ if it has the probability density function

$$f(x) = \frac{1}{\Gamma(t)}\lambda^t x^{t-1} e^{-\lambda x}, x \geqslant 0$$

where $\Gamma(t)$ is the gamma function

$$\Gamma(t) = \int_0^\infty x^{t-1} e^{-x} dx$$

If $t = 1$ then X is Exponentially distributed with parameter λ. The cumulative distribution function is the regularised gamma function

$$F(x) = \int_0^x f(u)du = \frac{\gamma(t, \lambda x)}{\Gamma(t)}$$

where $\gamma(t, \lambda x)$ is the lower incomplete gamma function. If t is a positive integer, the CDF follows the series expansion

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$F(x) = e^{-\lambda x} \sum_{i=t}^{\infty} \frac{(\lambda x)^i}{i!}$$

The skewness $\frac{2}{\sqrt{t}}$ depends only on the shape parameter t and approaches a normal distribution when t is large (when $t > 10$).

24.4.2 The Gaussian distribution

24.4.2.1 Linear

Linear combinations of jointly normal random variables are jointly normals, and independent normal random variables are jointly normal. Thus,

1. We can create jointly normal random variables from a set of independent normal random variables and take linear combinations.
2. Conversely, any set of jointly normal random variables can be reduced to linear combinations of independent normal random variables.

We provide an example for the case (2) in the case of a pair of correlated normal random variables (see Shreve [2004]). Let X_i, for $i = 1, 2$, be normally distributed with mean μ_i and variance σ_i^2. The correlation between X_1 and X_2 is ρ. Define

$$W = X_2 - \alpha X_1 \, , \alpha = \frac{\rho \sigma_2}{\sigma_1}$$

so that X_1 and W are independent.
Proof: It suffices to show that the covariance between X_1 and W is zero.
First,

$$E[W] = \mu_W = E[X_2] - \alpha E[X_1] = \mu_2 - \alpha \mu_1$$

Then,

$$\begin{aligned} \big(W - E[W]\big) &= \big((X_2 - E[X_2]) - \alpha(X_1 - E[X_1])\big) \\ &= (A - \alpha B) \end{aligned}$$

We compute the covariance between X_1 and W as

$$\begin{aligned} Cov(X_1, W) &= E[(X_1 - E[X_1])(W - E[W])] \\ &= E[AB] - \alpha E[B^2] \\ &= Cov(X_1, X_2) - \alpha \sigma_1^2 = 0 \end{aligned}$$

The variance of W is computed as follow

$$\begin{aligned} Var(W) = \sigma_W^2 &= E[(W - E[w])^2] \\ &= E[A^2] - 2\alpha E[AB] + \alpha^2 E[B^2] \\ &= (1 - \rho^2)\sigma_2^2 \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The joint density of X_1 and W is

$$f_{X_1,W}(x,y) = \frac{1}{2\pi\sigma_1\sigma_W} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2} - \frac{(y-\mu_W)^2}{2\sigma_W^2}}$$

Further, we see that we have decomposed X_2 into the linear combination

$$X_2 = \alpha X_1 + W$$

where (X_1, W) is a pair of normal random variables.
Using the above equation, we can then compute the conditional expectation of X_2 with respect to X_1. Given the properties of linearity and independence, the conditional expectation becomes

$$E[X_2|X_1] = \alpha X_1 + E[W] = \alpha(X_1 - \mu_1) + \mu_2 \tag{24.4.24}$$

which is a random quantity, but $\sigma(X_1)$-measurable. Subtracting this quantity from the equation on X_2, we see that the error made by the estimator is

$$X_2 - E[X_2|X_1] = W - E[W]$$

24.4.3 The exponential law and the Poisson law

Definition 24.4.1 *Exponential law*
The exponential law with parameter $\lambda > 0$ has a density $f_\lambda(x) = \lambda e^{-\lambda x} I_{x>0}$ with respect to Lebesgue measure. If X is a random variable with exponential law, its cumulative distribution function is, for $t \in \mathbb{R}^+$, given by

$$\mathbb{P}(X \leqslant t) = 1 - e^{-\lambda t} \tag{24.4.25}$$

The characteristic property of the exponential law is the lack of memory, as stated in the following proposition.

***Proposition* 17** *Properties of the density*

1. *The exponential law is the unique density on $\mathbb{R}^+$ such that*

$$\forall t, s > 0\,, \mathbb{P}(X > t + s | X > s) = \mathbb{P}(X > t)$$

2. *The exponential law is the unique density such that there exists a constant λ satisfying*

$$\forall t > 0\,,\ \lim_{s \to 0} s^{-1}\mathbb{P}(X > t + s | X > s) = \lambda$$

Some of the properties of the exponential law are discussed in the following proposition.

***Proposition* 18** *Properties of the exponential law*

1. *If X has an exponential law with parameter λ, then $E[X] = \lambda^{-1}$ and $Var(X) = \lambda^{-2}$.*
2. *The Laplace transform $E[e^{-uX}]$ of an exponential of parameter λ is defined for $u > -\lambda$ and is given by*

$$\phi(u) = E[e^{-uX}] = \frac{\lambda}{\lambda + u}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

3. *The sum, X_n, of n independent exponential random variables of parameter λ has a Gamma law with parameters (n, λ):*

$$\mathbb{P}(X_n \in dt) = \frac{(\lambda t)^{n-1}}{(n-1)!}\lambda e^{-\lambda t} I_{t>0} dt$$

where $\Gamma(n) = (n-1)!$.
The cumulative distribution function of X_n is

$$\begin{aligned} \mathbb{P}(X_n \leqslant x) &= 1 - e^{-\lambda x}\big(1 + \frac{\lambda x}{1!} + ... + \frac{(\lambda x)^{n-1}}{(n-1)!}\big) \\ &= 1 - e^{-\lambda x}\sum_{k=0}^{n-1}\frac{(\lambda x)^k}{k!} \end{aligned} \tag{24.4.26}$$

and its Laplace transform is given, for $u > -\lambda$, by

$$E[e^{-uX_n}] = \big(\frac{\lambda}{\lambda + u}\big)^n$$

4. *The law of $(X_1, ..., X_n)$ has density $\lambda^n e^{-\lambda x_n} I_{0<x_1<..<x_n}$.*
5. *The law of $(X_1, ..., X_n)$ given $X_n = x$ is*

$$\mathbb{P}(X_1 \in dx_1, ..., X_n \in dx_n | X_n = x) = \frac{(n-1)!}{x^n} I_{0<x_1<...<x_n<x} dx_1 ... dx_n$$

Note, from the knowledge of the c.d.f. in Equation (24.4.26), we can deduce the following useful property:

$$\mathbb{P}(X_n \leqslant x) - \mathbb{P}(X_{n+1} \leqslant x) = e^{-\lambda x}\frac{(\lambda x)^n}{n!} \tag{24.4.27}$$

Definition 24.4.2 *Poisson law*
A random variable X with integer values has a Poisson law with parameter $\theta > 0$ if

$$\mathbb{P}(X = k) = e^{-\theta}\frac{\theta^k}{k!}$$

Note, this result is a consequence of Equation (24.4.27).

***Proposition* 19** *Properties of the Poisson law*
If X has a Poisson law with parameter $\theta > 0$, then

1. *for any $s \in \mathbb{R}$, $E[s^X] = e^{\theta(s-1)}$.*
2. *$E[X] = \theta$, $Var(X) = \theta$.*
3. *for any $u \in \mathbb{R}$, $E[e^{iuX}] = e^{\theta(e^{iu}-1)}$.*
4. *for any $u \in \mathbb{R}$, $E[e^{uX}] = e^{\theta(e^{u}-1)}$.*

Conversely, if X is a random variable such that one of the property (1), (3), (4) holds, then X has a Poisson law with parameter θ.

***Corollary* 7** *If X and Y are two independent random variables, with a Poisson law of parameter θ and μ, then $X+Y$ has a Poisson law with parameter $\lambda + \mu$.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

24.4.4 The stable distributions

The class of stable distributions was characterised by Paul Levy in his study of sums of independent identically distributed terms in the 1920's. It is a rich class of probability distributions allowing for skewness and heavy tails with many intriguing properties. We are first going to define stable distributions and explain the reason why they are called stable. We follow Ouyang [2006] and Nolan [2014]. Note, we omit the letter d above the equal sign or an arrow to indicate equality in distribution.

Definition 24.4.3 *X is called stable, if for any positive number a and b, there exists a positive number c and real number d, such that*

$$aX_1 + bX_2 = cX + d \tag{24.4.28}$$

where X_1 and X_2 are independent copies of X. If the equation holds for $d = 0$, then it is called strictly stable. If it $X = -X$, then it is called symmetric.

***Proposition* 20** *For any stable random variable X, there exists $\alpha \in (0, 2]$, such that numbers in Equation (24.4.28) satisfies*

$$a^\alpha + b^\alpha = c^\alpha$$

In the Gaussian case, $X \sim N(\mu, \sigma^2)$ and we get $c = \sqrt{a^2 + b^2}$ and $d = (a + b - c)\mu$ with $\alpha = 2$.

Definition 24.4.4 *X is called stable, if for any integer $n \geqslant 2$, there exists positive numbers c_n, and real numbers d_n, such that*

$$\sum_{i=1}^{n} X_i = c_n X + d_n \tag{24.4.29}$$

where X_i are independent copies of X.

If Equation (24.4.29) holds, then $c_n = n^{\frac{1}{\alpha}}$.

A sequence of random variables $X = \{X_i\}_{-\infty}^{\infty}$, for some fixed $\delta > 0$, define a series of transformations T_n, such that

$$(T_n X)_i = \frac{1}{n^\delta} \sum_{j=in}^{(i+1)n-1} X_j \, , n \geqslant 1$$

This is called re-normalisation group transformation, and $\{T_n\}$ is a semi-group and satisfies $T_n T_m = T_{mn}$.

***Proposition* 21** *An i.i.d. strictly α-stable random variable sequence is a fixed point for T_n at $\delta = \frac{1}{\alpha}$. For dependent stable sequences, it is still a fixed point for T_n, but the corresponding δ is a function of α and the dependence.*

Definition 24.4.5 *X is stable if it has a domain attraction, that is, there exists i.i.d. random variables $Y_1, Y_2, ...$, $d_n > 0$, $a_n \in \mathbb{R}$, such that*

$$\frac{1}{d_n} \sum_{i=1}^{n} Y_i + a_n = X \tag{24.4.30}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

If X is Gaussian and Y_i are i.i.d. with finite variance, then Equation (24.4.30) is just the Central Limit Theorem (CLT). When $d_n = n^1\alpha$, then Y is said to belong to the normal domain of attraction X. In general, we have $d_n = n^1\alpha h(n)$, where $h(x)$ is a slowly varying function at infinity [1] We are now going to define stable distributions in terms of characteristic functions.

Definition 24.4.6 *X is stable if there exists $0 < \alpha \leqslant 2$, $c \geqslant 0$, $-1 \leqslant \beta \leqslant 1$, and real number μ,such that the characteristic function for X is*

$$E[e^{i\theta X}] = \begin{cases} e^{-c^\alpha|\theta|^\alpha\left(1-i\beta(sign\theta)\tan\frac{\pi\alpha}{2}\right)+i\mu\theta} \text{ if } \alpha \neq 1 \\ e^{-c|\theta|\left(1+i\beta\frac{2}{\pi}(sign\theta)\ln\theta\right)+i\mu\theta} \text{ if } \alpha = 1 \end{cases}$$

where α is the stability index, and c, β, μ are unique, except when $\alpha = 2$, β is arbitrary. We denote that $X \sim S(\alpha, \beta, c, \mu)$.

Many properties can be derived from the characteristic functions.

Proposition 22 *If X_1 and X_2 are independent, and $X_i \sim S(\alpha, \beta_i, c_i, \mu_i)$, then $X_1 + X_2 \sim S(\alpha, \beta, c, \mu)$ with*

$$c = (c_1^\alpha + c_1^\alpha)^{\frac{1}{\alpha}}, \ \beta = \frac{\beta_1 c_1^\alpha + \beta_2 c_2^\alpha}{c_1^\alpha + c_1^\alpha}, \ \mu = \mu_1 + \mu_2$$

Proposition 23 *If $X \sim S(\alpha, \beta, c, \mu)$, $a \in \mathbb{R}$, then $X + a \sim S(\alpha, \beta, c, a + \mu)$.*

Thus, μ is a location parameter.

Proposition 24 *If $0 < \alpha < 2$, and $X \sim S(\alpha, \beta, c, 0)$, then $-X \sim S(\alpha, -\beta, c, 0)$.*

which means that β is a skewness parameter.

Definition 24.4.7 *$S(\alpha, \beta, c, \mu)$ is called skewed to the right (left) if $\beta > 0$ ($\beta < 0$), and it is called totally skewed to the right (left) if $\beta = 1$ ($\beta = -1$).*

Note, the skewness here does not refer to the support of the distribution, but to the parameters P and Q in the Levy measure.

Proposition 25 *$X \sim S(\alpha, \beta, c, \mu)$ is symmetric if and only if $\beta = \mu = 0$.*

Symmetric distributions have real characteristic functions.

Proposition 26 *$X \sim S(\alpha, \beta, c, \mu)$, with $\alpha \neq 1$, then X is strictly stable if and only if $\mu = 0$.*

Proposition 27 *$X \sim S(1, \beta, c, \mu)$, then X is strictly stable if and only if $\beta = 0$.*

Proposition 28 *If $X_i \sim S(\alpha, \beta, c, \mu)$, then*

$$\sum_{i=1}^n X_i = \begin{cases} n^{\frac{1}{\alpha}}X_1 + \mu(n - n^{\frac{1}{\alpha}}), \ \alpha \neq 1 \\ nX_1 + \frac{2}{\pi}c\beta n\ln n, \ \alpha = 1 \end{cases}$$

[1] $\lim_{x\to\infty}\frac{h(ux)}{h(x)} = 1$, for any $u > 0$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Proposition 29 *If X has distribution $S(\alpha,\beta,c,0)$ with $\alpha<2$, then there exists two i.i.d. random variables Y_1 and Y_2, with common distribution $S(\alpha,1,c,0)$, such that*

$$X=\begin{cases}(\frac{1+\beta}{2})^{\frac{1}{\alpha}}Y_1+(\frac{1-\beta}{2})^{\frac{1}{\alpha}}Y_2 \text{ if } \alpha\neq 1\\ (\frac{1+\beta}{2})Y_1+(\frac{1-\beta}{2})Y_2+c\big(\frac{1+\beta}{\pi}\ln\frac{1+\beta}{2}-\frac{1-\beta}{\pi}\ln\frac{1-\beta}{2}\big) \text{ if } \alpha=1\end{cases}$$

it means that those stable subordinators can be used as building blocks of more general stable random variables.

Proposition 30 *Let $X\sim S(\alpha,\beta,c,0)$ with $0<\alpha<2$. Then*

$$\begin{cases}\lim_{\lambda\to\infty}\lambda^\alpha P(X>\lambda)=C_\alpha\frac{1+\beta}{2}c^\alpha\\ \lim_{\lambda\to\infty}\lambda^\alpha P(X<-\lambda)=C_\alpha\frac{1-\beta}{2}c^\alpha\end{cases}$$

where

$$C_\alpha=\big(\int_0^\infty x^{-\alpha}\sin x dx\big)^{-1}=\begin{cases}1-\alpha\Gamma(2-\alpha)\cos\frac{\pi\alpha}{2} \text{ if } \alpha\neq 1\\ \frac{2}{\pi} \text{ if } \alpha=1\end{cases}$$

Proposition 31 *Let $X\sim S(\alpha,\beta,c,0)$ with $0<\alpha<2$. Then*

$$E[|X|^p]<\infty \text{ for any } 0<p<\alpha$$
$$E[|X|^p]=\infty \text{ for any } p>\alpha$$

Proposition 32 *Let $X\sim S(\alpha,\beta,c,\mu)$ with $0<\alpha\leqslant 2$. Then the location parameter μ equals to the mean.*

If X is symmetric α-stable (SαS), then $X\sim S(\alpha,0,c,0)$. Its characteristic function is

$$E[e^{i\theta X}]=e^{-c^\alpha|\theta|^\alpha}$$

When $c=1$, X is called standard, and when $\alpha=2$, a standard SαS distribution is $N(0,2)$.

Proposition 33 *Let $X\sim S(\alpha',0,c,0)$ with $0<\alpha'\leqslant 2$. If $0<\alpha<\alpha'$, if we choose a random variable $A\sim S(\frac{\alpha}{\alpha'},1,(\cos\frac{\pi\alpha}{2\alpha'})^{\frac{\alpha'}{\alpha}},0)$, and assume X and A are independent, then*

$$Z=A^{\frac{1}{\alpha'}}X\sim S(\alpha,0,c,0)$$

In particular, if $\alpha'=2$, for a zero mean Gaussian random variable X, and if A is a positive $\frac{\alpha}{2}$-stable random variable independent of X, then $Z=A^{\frac{1}{\alpha'}}X\sim S\alpha S$. That is, every S$\alpha$S random variable is conditionally Gaussian.

We can also express stable distributions in terms of random series. We let $N(t)$ be the number of arrivals within time $[0,t]$, the process $\{N_t\}_{t\geqslant 0}$ is a homogeneous Poisson process with rate λ, if the inter-arrival times $\tau_{i+1}-\tau_i$, $i\geqslant 1$ are independent exponentially distributed with mean $\frac{1}{\lambda}$.

Proposition 34 *Let $\{\tau_i\}$ denote the arrival times of a Poisson process with rate 1, and $\{R_i\}$ are i.i.d. random variables, independent of the sequence $\{\tau_i\}$. If the series*

$$\sum_{i=1}^\infty \tau^{-\frac{1}{\alpha}}R_i$$

converges a.s., then it converges to a strictly α-stable random variable.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

24.4.5 Power-law distributions

The class of skew distribution functions (or Yule distribution) appears in a wide range of empirical data, particularly data describing sociological, biological and economic phenomena (see Simon [1955]). In the statistical extreme value theory (EVT), the extremes and the tail regions of a sample of independent and identically distributed (i.i.d.) random variables converge in distribution to one of only three types of limiting laws (see Reiss et al. [1997], Embrechts et al. [1999])

1. Exponential decay : it decreases at a rate proportional to its current value. That is, $X_t = X_0 e^{-\lambda t}$ where λ is the exponential decay constant and $\tau = \frac{1}{\lambda}$ is the mean lifetime.
2. Power-law decay : one quantity varies as a power of another. Power laws are scale-invariant, that is, for $f(x) = ax^{-\alpha}$ we get $f(cx) = c^{-\alpha} f(x) \propto f(x)$. A power law $x^{-\alpha}$ has a well-defined mean over $x \in [1, \infty]$ only if $\alpha > 2$ and it has a finite variance only if $\alpha > 3$.
3. The behaviour of distributions with finite endpoint of their support.

24.4.5.1 Definitions

More generally, a power-law probability distribution is a distribution whose density function has the form

$$p(x) \propto L(x) x^{-\alpha}$$

where $\alpha > 1$ and $L(x)$ is a slowly varying function at infinity [2]. Thus, $p(x)$ is asymptotically scale invariant.
Heavy-tailed distributions are probability distributions whose tails are not exponentially bounded. That is, they have heavier tails than the exponential distribution. The distribution of a random variable X with distribution function F is said to have a heavy right tail if

$$\lim_{x \to \infty} e^{\lambda x} P(X > x) = \infty \text{ for all } \lambda > 0$$

A fat-tailed distribution is a probability distribution exhibiting large skewness or kurtosis relative to the normal distribution. Some fat-tailed distributions have power law decay in the tail of the distribution, but do not necessarily follow a power law everywhere. The distribution of a random variable X is said to have a fat tail if

$$P(X > x) \sim x^{-\alpha} \text{ as } x \to \infty \,,\, \alpha > 0$$

That is, if X has a probability density function $f_X(x)$, then

$$f_X(x) \sim x^{-(1+\alpha)} \text{ as } x \to \infty \,,\, \alpha > 0$$

Since such a power is always bounded below by the probability density function of an exponential distribution, fat-tailed distributions are always heavy-tailed. Hence, the universe of fat-tailed distributions can be indexed by their tail index α with $\alpha \in (0, \infty)$. A standard measure of fat tails is the kurtosis, which is the expected fourth power of a process given by $E[X_t^4]$. Events deviating from the mean by five or more standard deviations in the normal distribution have lower probability, thus meaning that in the normal distribution rare events can happen but are likely to be milder in comparison to fat-tailed distributions. On the other hand, fat-tailed distributions such as the Cauchy distribution (and all other stable distributions with the exception of the normal distribution) are examples of fat-tailed distributions having undefined sigma (the variance is not bounded).

24.5 Some discrete variables and their distributions

For details see text book by Grimmett et al. [1992].

[2] $\lim_{x \to \infty} \frac{L(ux)}{L(x)} = 1$, for any $u > 0$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

24.5.1 Some popular distributions

24.5.1.1 Multinomial distribution

Given $x_i \in \{0, ..., n\}$, we let $x = (x_1, ..., x_d)$ be a vector. The multinomial distribution is

$$P(x|\,\theta) = \frac{n!}{\prod_{i=1}^d x_i!} \prod_{i=1}^d \theta_i^{x_i}$$

with $n = \sum_{i=1}^d x_i$, $\sum_{i=1}^d \theta_i = 1$, and $\theta_i \geqslant 0$.
When $n = 1$ the multinomial distribution simplifies to

$$P(x|\,\theta) = \prod_{i=1}^d \theta_i^{x_i}$$

with $\sum_{i=1}^d \theta_i = 1$ and $\theta_i \geqslant 0$.

24.6 Statistical models

See textbooks by Cox [2006], Konishi et al. [2008], Freedman [2009], among others.

24.6.1 Bayesian statistics

24.6.1.1 Bayesian rules

We consider two discrete random variables X and Y with joint probability mass function (pmf) $f_{X,Y}(x, y)$ and their marginal pmfs $f_X(x)$ and $f_Y(y)$. We can deduce the Bayes rule for pmfs. Using the law of total probability, the conditional probability mass function of Y given $X = x$, denoted $f_{Y|X}(y|x)$ (see Definition (24.1.12)), we get

$$\begin{aligned} f_{X|Y}(x|y) &= \frac{f_{X,Y}(x,y)}{f_Y(y)} \\ &= \frac{f_{Y|X}(y|x)}{f_Y(y)} f_X(x) \\ &= \frac{f_{Y|X}(y|x)}{\sum_{x \in Range(X)} f_{X,Y}(x,y)} f_X(x) \end{aligned}$$

Using the chain rule, we get another version of Bayes rule

$$f_{X|Y}(x|y) = \frac{f_{Y|X}(y|x)}{\sum_{x \in Range(X)} f_X(x) f_{Y|X}(y|x)} f_X(x)$$

We now consider two continuous random variables X and Y. Given the marginal pmf $f_X(x)$ and the conditional pmf $f_{Y|X}(y|x)$, the Bayes rule for densities is as follows:

$$\begin{aligned} f_{X|Y}(x|y) &= \frac{f_{Y|X}(y|x)}{f_Y(y)} f_X(x) \\ &= \frac{f_{Y|X}(y|x)}{\int_{-\infty}^{\infty} f_{X,Y}(u,y)du} f_X(x) \\ &= \frac{f_{Y|X}(y|x)}{\int_{-\infty}^{\infty} f_X(u) f_{Y|X}(y|u)du} f_X(x) \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We now discuss the case of a discrete random variable and a continuous one. We let θ be a discrete random variable with pmf $f_\Theta(\theta)$. For each $\Theta = \theta$, such that $f_\Theta(\theta) \neq 0$, let Y be a continuous random variable with conditional pdf $f_{Y|\Theta}(y|\theta)$. The conditional pmf of Θ given Y can be defined as the limit

$$\begin{aligned} f_{\Theta|Y}(\theta|y) &= \lim_{\Delta y \to 0} \frac{P\big(\Theta = \theta,\ y < Y + \Delta y\big)}{P(y < Y \leqslant y + \Delta y)} \\ &= \lim_{\Delta y \to 0} \frac{f_\Theta(\theta) f_{Y|\Theta}(y|\theta)\Delta y}{f_Y(y)\Delta y} \\ &= \frac{f_{Y|\Theta}(y|\theta)}{f_Y(y)} f_\Theta(\theta) \end{aligned}$$

Thus, we can write the Bayes rule as

$$f_{\Theta|Y}(\theta|y) = \frac{f_{Y|\Theta}(y|\theta)}{\sum_{\theta \in Range(\Theta)} f_\Theta(\theta) f_{Y|\Theta}(y|\theta)} f_\Theta(\theta)$$

24.6.1.2 Bayesian analysis

If a subset A of the variables $X_1, ..., X_n$ is conditionally dependent given another subset B of these variables, then the joint probability distribution $P(X_1, ..., X_n)$ is equal to $P(B) \cdot P(A|B)$. Therefore, it can be efficiently represented by the lower-dimensional probability distributions $P(B)$ and $P(A|B)$. Such conditional independence relations can be represented with a Bayesian network or copula functions.
Given some data x and parameters θ, a simple Bayesian analysis starts with a **prior probability** (or just prior), $p(\theta)$, and a likelihood function, $p(x|\ \theta)$, to compute a **posterior probability**

$$p(\theta|x) \propto p(x|\theta)p(\theta)$$

Often, the prior on θ depends in turn on other parameters φ that are not mentioned in the likelihood function. As a result, the prior $p(\theta)$ must be replaced by a likelihood $p(\theta|\varphi)$, and a prior $p(\theta|\varphi)$ on the newly introduced parameters φ is required, resulting in a posterior probability

$$p(\theta, \varphi|x) \propto p(x|\theta)p(\theta|\varphi)p(\varphi)$$

This is a simple example of a **hierarchical Bayes model**. The process may be repeated. For instance, the parameters φ may depend in turn on additional parameters ψ, which require their own prior. Eventually the process must terminate, with priors that do not depend on unmentioned parameters.
As an example, we consider the observations $x_1, ..., x_n$, each with normally distributed errors of known standard deviation σ. That is, $x_i \sim N(\theta_i, \sigma^2)$. We are interested in estimating the parameters θ_i, $i = 1, ..., n$. Assuming that the observations are independent, we could use a maximum likelihood approach. In that setting, the likelihood factorises and the maximum likelihood estimate is simply $\theta_i = x_i$. However, in the case where the quantities are related, say the individual parameters θ_i have themselves been drawn from an underlying distribution, then this relationship destroys the independence and suggests a more complex model. We could consider

$$x_i \sim N(\theta_i, \sigma^2) \text{ and } \theta_i \sim N(\varphi, \tau^2)$$

with improper priors $\varphi \sim flat$. When $n \geqslant 3$, we get an identified model (there exists a unique solution for the parameters of the model), and the posterior distributions of the individual θ_i will tend to move, or shrink away from the maximum likelihood estimates towards their common mean. This shrinkage is a typical behaviour in hierarchical Bayes models.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

24.6.2 Simple random walk

One of the simplest random processes is the simple random walk. At any time a particle inhabits one of the integer points of the real line. At time 0 it starts from point a, and at each subsequent epoch of time $1, 2, ...$ it moves from its current position to a new one according to a law. With probability p it moves one step to the right and with probability $q = 1-p$ it moves one step to the left. The moves are independent of each other. The walk is symmetric if $p = q = \frac{1}{2}$. Let S_n be the position of the particle after n moves, and set $S_0 = a$. Then,

$$S_n = a + \sum_{i=1}^{n} X_i$$

where $X_1, X_2, ...$ is a sequence of independent Bernoulli variables taking values $+1$ and -1 with probability p and q. The path of a particle is the collection of points in the plane, joined by solid lines between neighbours. The sequence $\{(n, S_n); n \geqslant 0\}$ has some important properties.

Lemma 24.6.1 *The simple random walk is spatially homogeneous, that is,*

$$P(S_n = j|S_0 = a) = P(S_n = j + b|S_0 = a + b)$$

Lemma 24.6.2 *The simple random walk is temporally homogeneous, that is,*

$$P(S_n = j|S_0 = a) = P(S_{n+m} = j|S_m = a)$$

Lemma 24.6.3 *The simple random walk has the Markov property, that is,*

$$P(S_{n+m} = j|S_0, S_1, ..., S_n) = P(S_{n+m} = j|S_n)$$

Thus, conditional upon knowing the value of the process at the nth step, its values after the nth step do not depend on its values before the nth step.

***Remark* 24.6.1** *Statements such as $P(S = j|X, Y) = P(S = j|X)$ are to be interpreted as meaning $P(S = j|X = x, Y = y) = P(S = j|X = x)$ for all x and y. This is a slight abuse of notation.*

24.6.3 Some Markov properties

Let $\{X_0, X_1, ...\}$ be a sequence (see Appendix (23.2.1.7)) of random variables taking values in some countable set S. Each X_n is a discrete random variable (d.r.v.) taking one of N possible values, where $N = |S|$.

Definition 24.6.1 *The process X is a Markov chain if it satisfies the Markov condition*

$$P(X_n = s|X_0 = x_0, X_1 = x_1, ..., X_{n-1} = x_{n-1}) = P(X_n = s|X_{n-1} = x_{n-1}) \tag{24.6.31}$$

for all $n \geqslant 1$ and all $s, x_0, x_1, ..., x_{n-1} \in S$.

This definition assume that both conditional probabilities are well defined, that is, $P(X_0 = x_0, X_1 = x_1, ..., X_{n-1} = x_{n-1}) > 0$.

***Remark* 24.6.2** *Markov chains can be described by a sequence of directed graphs, where the edges of graph n are labelled by the probabilities of going from one state at time n to the other states at time $n + 1$, denoted $P(X_{n+1} = x|X_n = x_{n+1})$. The same information is represented by the transition matrix from time n to time $n + 1$.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Equivalently, for each $s \in S$ and for every sequence $\{x_i; i \geqslant 0\}$ in S, we have

$$P(X_{n+1} = s|X_{n_1} = x_{n_1}, X_{n_2} = x_{n_2}, ..., X_{n_k} = x_{n_k}) = P(X_{n+1} = s|X_{n_k} = x_{n_k}) \text{ for all } n_1 < n_2 < ... < n_k \leqslant n \tag{24.6.32}$$

and

$$P(X_{n+m} = s|X_0 = x_0, X_1 = x_1, ..., X_n = x_n) = P(X_{n+m} = s|X_n = x_n) \text{ for any } m, n \geqslant 0$$

Since S is a countable set, it can be put in a one-to-one correspondence with some subset $S^{'}$ of the integers. For simplicity of exposition we let S be the set $S^{'}$ of integers. Thus, if $X_n = i$, the chain is in the ith state at the nth step. The evolution of the chain is described by its transition probability

$$P(X_{n+1} = j|X_n = i)$$

which depends the three quantities n, i, j. We will only consider chains depending upon i and j, that is, homogeneous.

Definition 24.6.2 *The chain X is called homogeneous if*

$$P(X_{n+1} = j|X_n = i) = P(X_1 = j|X_0 = i)$$

for all n, i, j. The transition matrix $P = (p_{ij})$ is the $|S| \times |S|$ matrix of transition probabilities

$$p_{ij} = P(X_{n+1} = j|X_n = i)$$

Thus, we let X be a Markov chain with transition matrix P.

Theorem 24.6.1 *P is a stochastic matrix, which is to say that*

1. *P has non-negative entries, or $p_{ij} \geqslant 0$.*
2. *P has row sums equal to one, or $\sum_j p_{ij} = 1$.*

In the short term the random evolution of X is described by P, while in the long term

Definition 24.6.3 *The n-step transition matrix $P_n = (p_{ij}(n))$ is the matrix of n-step transition probabilities*

$$p_{ij}(n) = P(X_{m+n} = j|X_m = i)$$

Note, $P_1 = P$.

Theorem 24.6.2 *Chapman-Kolmogorov equation*

$$p_{ij}(m+n) = \sum_k p_{ik}(m)p_{kj}(n)$$

Hence $P_{m+n} = P_m P_n$ and so $P_n = P^n$, the nth power of P.

Proof:

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}
p_{ij}(m+n) &= P(X_{m+n}=j|X_0=i) \\
&= \sum_k P(X_{m+n}=j, X_m=k|X_0=i) \\
&= \sum_k P(X_{m+n}=j|X_m=k, X_0=i)P(X_m=k|X_0=i) \\
&= \sum_k P(X_{m+n}=j|X_m=k)P(X_m=k|X_0=i)
\end{aligned}$$

as required. In the third line we have used the following result

$$P(A\cap B|C) = P(A|B\cap C)P(B|C)$$

and, in the fourth line, the Markov property in Equation (24.6.32).
This theorem relates the long-term development to the short-term one by telling us how X_n depends on the initial variable X_0. The rest follows from the definition of matrix multiplication.
We let $\mu_i^{(n)} = P(X_n = i)$ be the mass function (density) of X_n, and write $\mu^{(n)}$ for the row vector with entries $(\mu_i^{(n)}; i \in S)$.

Lemma 24.6.4

$$\mu^{(m+n)} = \mu^{(m)}P_n$$

and hence $\mu^{(n)} = \mu^{(0)}P^n$.

Proof:

$$\begin{aligned}
\mu_j^{(m+n)} &= P(X_{m+n}=j) = \sum_i P(X_{m+n}=j|X_m=i)P(X_m=i) \\
&= \sum_i \mu_i^{(m)} p_{ij}(n) = (\mu^{(m)}P_n)_j
\end{aligned}$$

and the result follows from Theorem (24.6.2).
Thus, the random evolution of the chain is determined by the transition matrix P and the initial mass function $\mu^{(0)}$. Hence, the study of the chain is largely reducible to the study of algebraic properties of matrices.

24.6.3.1 Some examples

24.6.3.1.1 Gibbs measure The Gibbs measure is a probability measure frequently seen in many problems of probability theory and statistical mechanics. It is a generalisation of the canonical ensemble to infinite systems. The canonical ensemble gives the probability of the system X being in state x (equivalently, of the random variable X having value x) as

$$P(X=x) = \frac{1}{Z(\beta)}e^{-\beta E(x)}$$

where $E(x)$ is a function from the space of states to the real numbers. In physics applications $E(x)$ is interpreted as the energy of the configuration x. The parameter β is a free parameter (in physics, it is the inverse temperature). The normalising constant $Z(\beta)$ is the partition function. However, in infinite systems, the total energy is no longer a finite number and cannot be used in the traditional construction of the probability distribution of a canonical ensemble. Traditional approaches in statistical physics studied the limit of intensive properties as the size of a finite system

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

approaches infinity (the thermodynamic limit). When the energy function can be written as a sum of terms that each involve only variables from a finite subsystem, the notion of a Gibbs measure provides an alternative approach. Gibbs measures provided a framework to directly study infinite systems, instead of taking the limit of finite systems.
A measure is a Gibbs measure if the conditional probabilities it induces on each finite subsystem satisfy a consistency condition: if all degrees of freedom outside the finite subsystem are frozen, the canonical ensemble for the subsystem subject to these boundary conditions matches the probabilities in the Gibbs measure conditional on the frozen degrees of freedom. The Gibbs measure of an infinite system is not necessarily unique, in contrast to the canonical ensemble of a finite system, which is unique. The existence of more than one Gibbs measure is associated with statistical phenomena such as symmetry breaking and phase coexistence.

24.6.3.1.2 The Ising model An example of the Markov property can be seen in the Gibbs measure of the Ising model. The probability for a given spin σ_k to be in state s could, in principle, depend on the states of all other spins in the system. Thus, we may write the probability as

$$P(\sigma_k = s|\, \sigma_j, j \neq k)$$

However, in an Ising model with only finite-range interactions (for example, nearest-neighbour interactions), we actually have

$$P(\sigma_k = s|\, \sigma_j, j \neq k) = P(\sigma_k = s|\, \sigma_j, j \in N_k)$$

where N_k a neighbourhood of the site k. That is, the probability at site k depends only on the spins in a finite neighbourhood. This last equation is in the form of a local Markov property. Measures with this property are sometimes called Markov random fields. More strongly, the converse is also true: any positive probability distribution (nonzero density everywhere) having the Markov property can be represented as a Gibbs measure for an appropriate energy function. This is the Hammersley-Clifford theorem.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 25

Introduction to stochastic calculus

25.1 An overview

25.1.1 Some definitions

25.1.1.1 stochastic processes

A stochastic process is a family $(X_t)_{t\in[0,T]}$ of random variables indexed by time.

Definition 25.1.1 *A stochastic process is a parametrised collection of random variables $\{X_t; t \in [0,T]\}$ defined on a probability space $(\Omega, \mathcal{F}, P)$ and assuming values in $\mathbb{R}^n$.*

For each realisation of the randomness ω, the trajectory $X_\cdot(\omega) : t \to X_t(\omega)$ defines a function of time called the sample path of the process. It can also be seen as function $X : [0,T] \times \Omega \to E$ of both time t and the randomness ω. We sometimes write $X(t,\omega)$ instead of $X_t(\omega)$ so that we may consider the process as a function of two variables

$$(t,\omega) \to X(t,\omega)$$

from $[0,T] \times \Omega$ into $\mathbb{R}^n$. Since events become less uncertain as more information becomes available through time the filtration describes how information is revealed progressively. When working with discontinuous functions we need to consider the class of càdlàg functions.

Definition 25.1.2 *Càdlàg function*
A function $f : [0,T] \to \mathbb{R}^d$ is said to be càdlàg if it is right-continuous with left limits. For each $t \in [0,T]$ the limits $f(t_-) = \lim_{s\to t, s<t} f(s)$ and $f(t_+) = \lim_{s\to t, s>t} f(s)$ exist and $f(t) = f(t_+)$.

Any continuous function is càdlàg but càdlàg functions can have discontinuities. If t is a discontinuity point we denote by

$$\Delta f(t) = f(t) - f(t_-)$$

the jump of $f(\cdot)$ at time t. Càdlàg function on $[0,T]$ has a finite number of large jumps and a possibly infinite but countable number of small jumps. We can consider as an example a step function having jumps at points t_i whose values at t_i are defined to be the value after the jump, that is $f = I_{[t_i,t_{i+1}[}$ and such that $f(t_i) = f(t_{i+})$. If we had defined $f(t_i)$ to be the value before the jump $f(t_i) = f(t_{i-})$ we would have obtained a left-continuous function with right limits called càdlàg. In fact, if a right-continuous function has a jump at time t then the discontinuity is seen as a sudden event while if the sample paths were left-continuous an observer approaching t along the path could predict the value at time t. Therefore, if we want to model a discontinuous process whose values are predictable we should

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

use a càdlàg process, while if we let the jumps represent sudden events we should use a càdlàg process. The space of càdlàg function is the Skorokhod space denoted by $D([0,T])$. We saw that to add some time-dependent ingredient to the structure of the probability space we consider the notion of filtration.

Definition 25.1.3 *A filtration or information flow on $(\Omega, \mathcal{F}, \mathbb{P})$ is an increasing family of σ-algebras $(\mathcal{F}_t)_{t\in[0,T]}$, that is $\forall t \geqslant s \geqslant 0$, $\mathcal{F}_s \subseteq \mathcal{F}_t \subseteq \mathcal{F}$.*

An $\mathcal{F}_t$-measurable random variable is a random variable whose value will be revealed at time t and a process whose value at time t is revealed by the information $\mathcal{F}_t$ is said to be nonanticipating or adapted.

Definition 25.1.4 *A stochastic process $(X_t)_{t\in[0,T]}$ is said to be nonanticipating with respect to the information structure $(\mathcal{F}_t)_{t\in[0,T]}$ or $\mathcal{F}_t$-adapted if for each $t \in [0,T]$ the value of X_t is revealed at time t (that is the random variable X_t is F_t-measurable).*

Given a stochastic process X and an information flow $(\mathcal{F}_t)_{t\in[0,T]}$ we now consider the process X as a function defined on $[0,T] \times \Omega$ with a σ-algebra generated by the process X.

Definition 25.1.5 *Optional process*
The optional σ-algebra is the σ-algebra $\mathcal{O}$ generated on $[0,T] \times \Omega$ by all nonanticipating càdlàg processes. A process $X : [0,T] \times \Omega \to \mathbb{R}^d$ which is measurable with respect to $\mathcal{O}$ is called an optional process.

There is also a σ-algebra on $[0,T] \times \Omega$ for left-continuous processes.

Definition 25.1.6 *Predictable process The predictable σ-algebra is the σ-algebra $\mathcal{P}$ generated on $[0,T] \times \Omega$ by all nonanticipating left-continuous processes. A process $X : [0,T] \times \Omega \to \mathbb{R}^d$ which is measurable with respect to $\mathcal{P}$ is called an predictable process.*

From the definitions above, all predictable processes are generated from left-continuous processes, while any nonanticipating process with càdlàg trajectories is optional.

***Remark* 25.1.1** *In the financial world we let state variables such as market prices be modelled as optional processes, while anything to do with investors decisions will be represented by predictable processes.*

Definition 25.1.7 *Let X be a càdlàg process. We denote by $\mathcal{F}^X$ its natural filtration. The process is said to have*

- *independent increments if, for any pair (s,t) of positive numbers, $X_{t+s} - X_t$ is independent of $\mathcal{F}_t^X$.*
- *stationary increments if, for any pair (s,t) of positive numbers, $X_{t+s} - X_t \overset{\text{law}}{=} X_s$.*

25.1.1.2 Local martingales and semi-martingales

Definition 25.1.8 *Local martingale*
An adapted, right-continuous, process M is an F-local martingale if there exists a sequence of stopping times (T_n) such that

1. *The sequence T_n is increasing and $\lim_{n\to\infty} T_n = \infty$, a.s..*
2. *For every n, the stopped process $M^{T_n} I_{T_n>0}$ is an F-martingale.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

A sequence of stopping times such that (1) holds is called a localising or reducing sequence.

Given a process $(X_t)_{t\geqslant 0}$ with stationary independent increments (possibly with jumps), it is often useful to compensate it, that is, subtract the average claim size $E[X_t]$. Assuming that its probability law has finite expectation, and $E[X_t] = \gamma t$, we get the representation

$$X_t = \gamma t + \big(X_t - E[X_t]\big)$$

The compensated process $\big(X_t - E[X_t]\big)$ is a martingale, such that X_t can be decomposed in a linear drift γt and a martingale. Thus, we can define a semi-martingale as a process which is adapted to a filtration $(\mathcal{F}_t)_{t\geqslant 0}$, has right-continuous paths, and has left limits (càdlàg paths) and allows a decomposition

$$X_t = X_0 + V_t + M_t \,,\, t \geqslant 0 \tag{25.1.1}$$

where $V = (V_t)_{t\geqslant 0}$ is an F-adapted càdlàg process of finite variation and $M = (M_t)_{t\geqslant 0}$ is an F-local martingale. In general, this decomposition is not unique, and one must add some conditions on the finite variation process to get uniqueness. For instance, in presence of jumps, the above representation is unique if we consider special semimartingales where the big jumps have been taken away (for example, jumps with absolute jump size bigger than 1). This is because a semimartingale with bounded jumps is special (see Jacod et al. [1987]). Let $\Delta X_t = X_t - X_{t-}$ be the jump at time t if there is any. Then

$$X_t - \sum_{s\in[0,t]} \Delta X_s I_{|\Delta X_s|>1} \tag{25.1.2}$$

has bounded jumps.

Definition 25.1.9 *A special semi-martingale is a semi-martingale with a predictable finite variation part. Such a decomposition (see Equation (25.1.1)) with V predictable, is unique. It is called the canonical decomposition of X, if it exits.*

That is, any F-local martingale M (with $M_0 = 0$) admits a unique orthogonal decomposition into a local martingale with continuous paths M^c and a purely discontinuous, local martingale M^d. Assuming $X_0 = 0$, we get the following unique representation for semi-martingales

$$X_t = V_t + M^c + M^d + \sum_{s\in[0,t]} \Delta X_s I_{|\Delta X_s|>1}$$

such that $X_t - \sum_{s\in[0,t]} \Delta X_s I_{|\Delta X_s|>1}$ satisfies Equation (25.1.1).

25.1.1.3 The quadratic variation

25.1.1.3.1 The covariation of martingales Recall, for square integrable random variables, we consider the space $L_2(\mathbb{R})$ on the probability space $(\Omega, \mathcal{F}, P)$. This is the Hilbert space with scalar product $< X, Y >$. We are now going to define the covariation of the martingales X and Y.

1. Continuous local martingales: Let X be a continuous local martingale. The predictable quadratic variation process of X is the continuous increasing process $< X >$ such that $X^2 - < X >$ is a local martingale. Let X and Y be two continuous local martingales. The predictable covariation process is the continuous finite variation process $< X, Y >$ such that $XY - < X, Y >$ is a local martingale. In the special case where X, Y are Gaussian martingales, we get $< X, Y >= E[XY]$. Note, $< X >=< X, X >$ and

$$< X + Y >=< X > + < Y > +2 < X, Y >$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. General local martingales: Let X and Y be two local martingales. The covariation process is the finite variation process $[X,Y]$ such that $XY-[X,Y]$ is a local martingale. Further, $\Delta[X,Y]_t = \Delta X_t \Delta Y_t$. The process $[X]=[X,X]$ is non-decreasing. If the martingales X and Y are continuous, $[X,Y]=<X,Y>$. If $\mathbb{P}$ and $\mathbb{Q}$ are equivalent probability measures, the quadratic covariation process $[X,Y]$ under $\mathbb{P}$ and under $\mathbb{Q}$ are the same. The covariation $[X,Y]$ of both processes X and Y can be also defined by polarisation

$$[X+Y]=[X]+[Y]+2[X,Y]$$

3. Semi-martingales: If X and Y are semi-martingales and if X^c, Y^c are their continuous martingale parts, their quadratic covariation is

$$[X,Y]_t =< X^c, Y^c >_t + \sum_{s\leqslant t}(\Delta X_s)(\Delta Y_s)$$

The integration by parts formula is

$$d(X_tY_t) = X_{t^-}dY_t + Y_{t^-}dX_t + d[X,Y]_t \tag{25.1.3}$$

Using the integration by parts formula for semi-martingales we can define the quadratic variation.

Definition 25.1.10 *The quadratic variation process of a semi-martingale X on $[0,t]$ is the nonanticipating cadlag process defined by*

$$[X,X]_t = |X_t|^2 - 2\int_0^t X_{s_}dX_s$$

The quadratic variation process is a limit in probability. If $\pi^n = \left(t_0^n = 0 < t_1^n < ... < t_{n+1}^n = T\right)$ is a sequence of partitions of $[0,T]$ such that $|\pi^n| = \sup_k |t_k^n - t_{k-1}^n| \to 0$ as $n\to\infty$ then

$$\sum_{\substack{t_i\in\pi^n\\ 0\leqslant t_i<t}}\left(X_{t_{i+1}}-X_{t_i}\right)^2 \underset{n\to\infty}{\overset{\mathbb{P}}{\to}} [X,X]_t$$

where the convergence is uniform in t. We can then deduce that the quadratic variation $[X,X]_t$ is an increasing process, and we can define the integral $\int_0^t \phi d[X,X]$. Also, its jumps are related to the jumps of X by $\Delta[X,X]_t = |\Delta X_t|^2$.

***Remark* 25.1.2** *The quadratic variation contrarily to the variance is not defined by taking expectations. It is a random process.*

From the Definition (25.1.5) the process $[X,X]_t$ is an optional process while from Definition (25.1.6) the process $<X,X>_t$ is a predictable process.

25.1.1.3.2 Some examples We are now going to give some examples of the quadratic variation for some specific processes X. We let $U_t = \int_0^t \sigma(\omega,s)dW_s$, where W_t is a Brownian motion, be the stochastic integral with quadratic variation

$$[U,U]_t =< U,U >_t = \int_0^t \sigma^2(\omega,s)ds$$

In this special case, the quadratic variation $[U,U]_t$ is equal to the variance of U_t. This is why the market calls it the realised variance.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We consider the jump-diffusion process $(X_t)_{t\geqslant 0}$ in $\mathbb{R}$ made of a Brownian term, which is a stochastic integral, and a compound jump process, so that its quadratic variation is

$$[X,X]_t = \int_0^t \sigma^2(\omega,s)ds + \sum_{\substack{s\in[0,t]\\ \Delta X_s\neq 0}} |\Delta X_s|^2 = \int_0^t \sigma^2(\omega,s)ds + \int_{[0,t]}\int_{\mathbb{R}} z^2 J_X(\omega, ds\times dz)$$

25.1.2 Basic concepts

A stochastic process is a family of random variables defined on a common probability space, indexed by the elements of an ordered set T, called the parameter set. In general, T is an interval of time and the random variable indexed by an element $t\in T$ describe the state of the process at time t.

Definition 25.1.11 *A stochastic process is a family of (real or complex valued) random variables $\{X(t); t\in T\}$, defined on a common probability space $(\Omega,\mathcal{B},P)$, where the parameter set T is a subset of the real line $\mathbb{R}$.*

The parameter set T is often called the domain of definition of the stochastic process $X(t)$. We will consider two cases

1. if $T = N_+ = \{0,1,...\}$ the process is a discrete parameter stochastic process.
2. if T is an interval of the real line, the process is said to have a continuous parameter.

In general, T will either be N_+ or $\mathbb{R}_+ = [0,\infty)$.
Let $\{X(t); t\in T\}$ be a real-valued random process (r.r.p.) and $\{t_1,...,t_n\}\subset T$, where $t_1<t_2<...<t_n$, then

$$F_{t_1,...,t_n}(x_1,...,x_n) = P\big(X(t_1)\leqslant x_1,...,X(t_n)\leqslant x_n\big) \tag{25.1.4}$$

is a finite-dimensional marginal distribution function of the process $\{X(t); t\in T\}$. One of the basic characteristics of a r.r.p. are its marginal distribution functions.

Definition 25.1.12 *We shall say that the random process $\{X(t); t\in T\}$ is given if all its marginal distribution functions*

$$\{F_{t_1,...,t_n}(\cdot,...,\cdot)\}$$

obtained for all finite subsets $\{t_1,...,t_n\}\subset T$ are given.

The marginal distribution functions defined above must satisfy the following consistency conditions:

1. $F_{t_{k_1},...,t_{k_n}}(x_{k_1},...,x_{k_n}) = F_{t_1,...,t_n}(x_1,...,x_n)$ for any permutation $k_1,...,k_n$ of $1,...,n$.
2. for any $1\leqslant k<n$ and $x_1,...,x_n\in\mathbb{R}$

$$F_{t_1,...,t_k}(x_1,...,x_k) = F_{t_1,...,t_n}(x_1,...,x_k,\infty,...,\infty)$$

From Definition (25.1.11), for each $t\in T$ fixed, $X(t) = X(t,\cdot)$ is a random variable defined on Ω, while for any $\omega\in\Omega$ fixed, $X(\cdot,\omega)$ is a function defined on the parameter set T. A stochastic process (say, real) $\{X(t); t\in T\}$ is said to be directly given if the sample space is $\Omega = \mathbb{R}^T$, where $\mathbb{R}^T$ is the set of all functions on the parameter set T with values in $\mathbb{R}$. That is, any $\omega\in\Omega$ is a mapping

$$\omega; T\to\mathbb{R}$$

The stochastic process $\{X(t); t\in T\}$ is then defined as a family of coordinate mappings on $\mathbb{R}^T$ to $\mathbb{R}$. Thus, for any $t\in T$, $X(t,\cdot)$ is a random variable on $\mathbb{R}^T$ defined as

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\text{for any } \omega(\cdot) \in \mathbb{R}^T \text{ , } X(t,\omega) = \omega(t)$$

For any time interval T, we define a real random function (r.r.f.) as

$$(t,\omega) \to X(t,\omega) \text{ , } t \in T \subset \mathbb{R} \text{ , } \omega \in \Omega$$

such that

1. the trajectories $t \to X(t,\omega)$ are continuous on T
2. the values $\omega \to X(t,\omega)$ are random variables on Ω

It is convenient to introduce the concept of state space $\{S, \mathscr{L}\}$ of a process where S is the set of values of the process and $\mathscr{L}$ is a σ-algebra of subsets of S. For instance, if the process is real, S will be the real line or part of it, and $\mathscr{L}$ will be the σ-field of Borel subsets of S.
We define the n-dimensional Borel cylinder as follow: let $B_1, B_2, ..., B_n$ be n arbitrary Borel subsets of the real line, and consider the following subset of $\mathbb{R}^T$:

$$\{\omega; X(t_1) \in B_1, ..., X(t_n) \in B_n\}$$

where $\{t_1, ..., t_n\} \subset T$ and $X(t,\cdot)$ is coordinate mapping on $\mathbb{R}^T$ to $\mathbb{R}$. Then the σ-algebra $\mathcal{B}$ is defined as the least σ-algebra of subsets of $\mathbb{R}^T$ containing all the finite-dimensional Borel cylinders.
The problem with this σ-field is that the sample space $\Omega = \mathbb{R}^T$ is too big and the σ-algebra $\mathcal{B}$ is too small, in the sense that many important subsets of Ω are not members of $\mathcal{B}$. Thus, we can not compute their probabilities since the probability measure P is defined on $\mathcal{B}$. For example, the subset of all continuous functions on $\mathbb{R}^T$ is not an event (that is, a member of $\mathcal{B}$). The subset

$$\{\omega; \sup_{t \in T} X(t) \leqslant x\} = \bigcap_{t \in T} \{X(t) \leqslant x\}$$

does not belong in general to $\mathcal{B}$. In fact, any subset of $\mathbb{R}^T$ that is an intersection or union of uncountably many events from $\mathcal{B}$ may not necessarily be an element of $\mathcal{B}$. Such problems do not arise when T is a countable set. These difficulties were alleviated by the separability concept introduced by Doob (see Doob [1937], [1953]). Between these two dates, Doob developed further the theory and a detailed description can be found in Getoor [2009].
Kolmogorov [1931] [1933] showed how to construct a family $\{X(t); t \in T\}$ of random variables with given finite dimensional distributions where T is an arbitrary index set. Assuming T to be countable, the Kolmogorov construction provided a mathematical model for a sequence of independent random variables with given distributions. We give a description later in this section. The intuition being that the finite dimensional distributions associated with an empirical process are (in theory) the only things being physically observable. Thus, any family of random variables with the given finite dimensional distributions ought to be a reasonable model for the underlying empirical process. However, this is no longer the case when T is uncountable. In our example above, we saw that the probabilities that $t \to X_t$ is continuous or that $t \to X_t$ is measurable are not defined in such a construction. This is the problem addressed by Doob [1937] and solved using measure theory. He distinguished the mathematical object from the empirical process

Before discussing the concept of separability, we need to introduce the notion of equivalence for stochastic processes. Let $\{X(t); t \in T\}$ and $\{Y(t); t \in T\}$ be two stochastic processes defined on a common probability space $(\Omega, \mathcal{B}, P)$ and taking values in the same state space $\{S, \mathscr{L}\}$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 25.1.13 *If, for each $n = 1, 2, ...$*

$$P\big(X(t_1) \in B_1, ..., X(t_n) \in B_n\big) = P\big(Y(t_1) \in B_1, ..., Y(t_n) \in B_n\big)$$

for all $\{t_1, ..., t_n\} \subset T$ and $\{B_1, ..., B_n\} \subset \mathscr{L}$, the random processes $X(t)$ and $Y(t)$ are said to be stochastically equivalent in the wide sense.

Definition 25.1.14 *If, for every $t \in T$,*

$$P\big(X(t) = Y(t)\big) = 1$$

the random processes are said to be stochastically equivalent or just equivalent.

Note, the latter definition implies the former one.

Definition 25.1.15 *Let $\{\xi(t); t \in T\}$ be a stochastic process on $(\Omega, \mathcal{B}, P)$ with state space $\{S, \mathscr{L}\}$. Any other stochastic process $\{\xi(t); t \in T\}$ on the same probability space, equivalent to $\xi(t)$, is called a version of $\xi(t)$.*

Definition 25.1.16 *The stochastic processes $\{X(t); t \in T\}$ and $\{Y(t); t \in T\}$ are said to be indistinguishable if almost all their sample paths coincide on T. In other words, if*

$$P\big(X(t) = Y(t)\,, \forall t \in T\big) = 1$$

This is the strongest form of equivalence, which clearly implies the other two. The main difference between the last two definition comes from the characterisation of the null set Λ_t for $X(t)$ and $Y(t)$. In the last definition, there is just one null event Λ such that

$$X(t, \omega) = Y(t, \omega) \text{ on } T$$

for every $\omega \in \Lambda^c$. The definitions are equivalent if T is a countable set. We need some conventions in the case of continuous time. Let $\{\xi(t); t \in T\}$ be a stochastic process with state space $\{S, \mathscr{L}\}$, which is a metric space. We call the process Cadlag if each of its trajectory is a right continuous function and has limits on the left.

***Proposition* 35** *Let $\{X(t); t \in T\}$ and $\{Y(t); t \in T\}$ be stochastically equivalent and both right continuous. Then $X(t)$ and $Y(t)$ are indistinguishable.*

We are now going to introduce the Kolmogorov construction (see Kolmogorov [1933]) as a way of creating a stochastic process starting from a consistent family of distribution functions. We let $\{X(t); t \in T\}$ be a stochastic process on $(\Omega, \mathcal{B}, P)$ with state space $\{\mathbb{R}, \mathcal{R}\}$, where $\mathbb{R}$ is the real line and $\mathcal{R}$ is the σ-algebra of Borel subsets of $\mathbb{R}$. The stochastic process determines a consistent family of marginal distribution functions given by Equation (25.1.4). A set in $\mathcal{F}$ depends on at most a countable number of t-values. The Kolmogorov theorem states that the converse is true.

Theorem 25.1.1 *Kolmogorov theorem*
Assume that

$$\{F_{t_1,...,t_n}(x_1, ..., x_n)\}\,, \{t_1, ..., t_n\} \subset T\,, n = 1, 2, ...$$

is a consistent family of distribution functions, then there exists a stochastic process with the above equation as its marginal distribution functions.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We let this stochastic process be defined on the probability space $(\mathbb{R}^T, \mathcal{B}, P)$, where T is an arbitrary index set, and specified by

$$X(t,\omega) = \omega(t) \text{ for each } \omega \in \mathbb{R}^T \text{ and } t \in T$$

The Kolmogorov space $(\Omega, \mathcal{B}, P)$ provides a satisfactory model for a sequence of random variables with prescribed finite dimensional distributions.
However, when requiring some regularity conditions on the sample functions, such as continuity and measurability, this is no-longer the case. That is, these events are not included in $\mathcal{F}$, and their P probabilities are not defined. We follow the separability approach introduced by Doob [1953] and denote this particular subset of $\mathbb{R}^T$ by Ω_0. In order for the process $\{\xi_0(t); t \in T\}$ to be stochastically equivalent to $\{\xi(t); t \in T\}$, which can be realised in Ω_0, it is necessary and sufficient that, if $\Omega_0 \subset \Omega$, $P^*(\Omega_0) = 1$, where P^* is the outer measure induced by P as follows: For any $M \subset \mathbb{R}^T$, define $P^*(M)$ as the infimum of $P(M^{'})$ for $M^{'} \in \mathcal{F}$ with $M^{'} \supset M$. That is,

$$P^*(M) = \inf\{P(M^{'}); M^{'} \supset M\}$$

where $M^{'} \supset \mathbb{R}^T$ is a cylinder set. If these conditions are satisfied, we can define a σ-algebra $\mathcal{F}_0$ on Ω_0 consisting of the sets $M_0 \subset \Omega_0$, such that $M_0 = \Omega_0 \cap M$ for some $M \in \mathcal{F}$, and set $P_0(M_0) = P(M)$. In such a case, Doob proved that P_0 is well defined and that the system $(\mathbb{R}^T, \mathcal{B}, P_0)$ is the new probability space with $\mathcal{B}_0 = \mathcal{B} \cap \Omega_0$.
Doob defined the concept of separability as follow:

Definition 25.1.17 *The stochastic process $\{X(t); t \in T\}$, where $T \subset \mathbb{R}$ is an interval, is said to be separable if there exists a countable dense subset $D \subset T$ and a null event $\Lambda \subset \Omega$ such that*

$$\{\omega; X(t,\omega) \in C, t \in l \cap D\} - \{\omega; X(t,\omega) \in C, t \in l \cap T\} \subset \Lambda$$

for any closed set C and any open interval l.

The countable set D is called the separant. Note that

$$\{\omega; X(t,\omega) \in C, t \in l \cap D\} \supset \{\omega; X(t,\omega) \in C, t \in l \cap T\}$$

From the above definition, we get

$$\Lambda^c \cap \{\omega; X(t,\omega) \in C, t \in l \cap D\} = \Lambda^c \cap \{\omega; X(t,\omega) \in C, t \in l \cap T\}$$

implying that the RHS is an event because the LHS is one.
The next proposition shows that every trajectory of the process $\{X(t); t \in T\}$ from Λ^c is as regular on T as its restriction on D.

Proposition 36 *For each $\omega \in \Lambda^c$ and open interval $l \subset T$,*

$$\overline{X(l \cap T, \omega)} = \overline{X(l \cap D, \omega)}$$

Note, $\overline{X(l \cap T, \omega)}$ denotes the closure in $\mathbb{R}$ of the set of values assumed by the trajectory $X(\cdot, \omega)$ as t varies in $l \cap T$. Thus, we obtain that for each $\omega \in \Lambda^c$ and $t \in T$,

$$X(t,\omega) = \overline{X(l \cap D, \omega)}$$

for every open interval l containing t. On the other hand, since D is dense in T, it implies that, for every $t \in T$, there exists a sequence $\{u_k\}_1^\infty \subset D$ such that $u_k \to t$. Then, for each $\omega \in \Lambda^c$,

$$\lim_{k \to \infty} X(u_k, \omega) = X(t, \omega)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where the sequence $\{u_k\}$ may depends on ω. Thus, the values of the sample functions on T are limits of sequences $\{X(u_k,\cdot)\}_1^\infty$. It can be seen as follows: let $t \in l$, then

$$\lim_{k\to\infty} X(u_k,\omega) = \overline{X(l \cap D,\omega)}$$

One can show that for every $\omega \in \Lambda^c$ and open set t,

$$\sup_l X(t,\omega) = \sup_{l\cap D} X(t,\omega)$$

In the same fashion, one can show that

$$\inf_l X(t,\omega) = \inf_{l\cap D} X(t,\omega)$$

In the following proposition Doob showed that separable processes exist.

Proposition 37 *Every real-valued stochastic process $\{X(t); t \in T\}$ defined on a complete probability space $(\Omega, \mathcal{B}, P)$ has a separable version.*

It implies that for any real random process $\{X(t); t \in T\}$ on a complete probability space $(\Omega, \mathcal{B}, P)$, there is a real random process $\{\tilde{X}(t); t \in T\}$ defined on the same probability space, which is separable and stochastically equivalent to $X(t)$. While $\tilde{X}(t)$ and $X(t)$ may have different trajectories, it is not the case if both are càdlàg (see Definition (25.1.2))

The set of sample functions is $\mathbb{R}^T$, and we usually choose a set with particular regularity conditions. The theory of stochastic processes is concerned with different features of such family, many of which can be studied by means of their marginal distributions. For simplicity of exposition, we are going to list five classes of random processes, each of them having a particular stochastic structure.

1. Processes with independent increments. Let $\{X(t); t \in T\}$ be a real-valued stochastic process on a probability space $\{\Omega, \mathcal{B}, P\}$ where $T \subset \mathbb{R}$ is an interval.

   Definition 25.1.18 *The process $X(t)$ is said to have independent increments if for any finite subset $\{t_0, ..., t_n\} \subset T$ with $t_0 < ... < t_n$, the increments*

   $$X(t_0), X(t_1) - X(t_0), ..., X(t_n) - X(t_{n-1})$$

   are independent random variables.

   Thus, all the marginal distributions are completely determined by the distribution of $X(t)$ for all $t \in T$ and by $X(t_2) - X(t_1)$, $t_1, t_2 \in T$, $t_1 < t_2$. Some examples are the Poisson and Wiener processes.

2. Markov processes. We consider the process $X(t)$ taking values in $\mathbb{R}$ and such that $T = [0, \infty)$.

   Definition 25.1.19 *The process $X(t)$ is called Markov with the state space $\{\mathbb{R}, \mathcal{R}\}$ if, for any $0 \leqslant t_1 < ... < t_n$ and $B \in \mathcal{R}$,*

   $$P(X(t_n) \in B | X(t_1), ..., X(t_{n-1})) = P(X(t_n) \in B | X(t_{n-1})) \text{ a.s.}$$

   If we fix $t \in (0, \infty)$ and consider the families of random variables $\{X(s); s \leqslant t\}$ and $\{X(u); u \geqslant t\}$, where t is the present, then for any $B_1, B_2 \in \mathcal{R}$ and $s < t < u$, the above definition implies

   $$P(X(s) \in B_1, X(u) \in B_2 | X(t)) = P(X(s) \in B_1 | X(t)) P(X(u) \in B_2 | X(t))$$

   Thus, the Markov property means that given the present, the future and the past of a Markov process are independent.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 25.1.20 *The distribution π on $\mathcal{R}$ defined by*

$$\pi(B) = P(X(0) \in B)$$

is called the initial distribution of the process.

Definition 25.1.21 *A version*

$$P(s, t, x, B)$$

of $P(X(t) \in B|X(s) = x)$ having the properties

(a) for every $s < t$ and $x \in \mathbb{R}$ fixed, $P(s, t, x, \cdot)$ is a probability measure on $\mathcal{R}$,

(b) for every $s < t$ and $B \in \mathcal{R}$ fixed, $P(s, t, \cdot, B)$ is an $\mathcal{R}$-measurable function,

is called the transition probability function (or simply the transition probability) of the Markov process.

From the definition, it follows that

$$P(t, t, x, B) = \begin{cases} 1 \text{ if } x \in B \\ 0 \text{ if } x \notin B \end{cases}$$

for all $t \geqslant 0$. Further, the transition probability satisfies the Chapman-Kolmogorov equation. For any $0 \leqslant s < t < u$ and $B \in \mathcal{R}$,

$$P(s, u, x, B) = \int_{\mathbb{R}} P(s, t, x, dy)P(t, u, y, B) \text{ , } 0 \leqslant s < t < u \tag{25.1.5}$$

The initial distribution π and the transition probability function $P(s, t, x, B)$ completely and uniquely determine the probability measure P of the process.

Definition 25.1.22 *A Markov process is said to be homogeneous or to have stationary transition probability if, for any $0 \leqslant s < t$,*

$$P(s, t, x, B) = P(0, t - s, x, B)$$

To simplify notation, we can write

$$P(x, t, B) = P(0, t, x, B)$$

and the Chapman-Kolmogorov equation becomes

$$P(x, s + t, B) = \int_{\mathbb{R}} P(x, s, dy)P(y, t, B) \tag{25.1.6}$$

Thus, the stationary transition probability corresponds to

$$P(x, t, B) = P(X(t) \in B|X(0) = x)$$

From the Equation (25.1.6), it is enough to know $P(x, t, B)$ for all $s \leqslant \epsilon$, where $\epsilon > 0$ is arbitrarily small, since, for all $u > \epsilon$ it is determined by the above equation. That is, the local behaviour of the process (in the neighbourhood of zero) determines its global behaviour.

3. Strictly stationary processes. This is the class of stochastic processes whose marginal distributions are invariant under time-shift.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 25.1.23 *A real-valued stochastic process $\{X(t); t \in T\}$ on a probability space $(\Omega, \mathcal{B}, P)$ is said to be strictly stationary if for any $\{t_1, ..., t_n\} \subset T$ then $\{t_{1+h}, ..., t_{n+h}\} \subset T$, and any $(B_k)_1^n \subset \mathcal{R}$, $n = 1, 2, ...,$*

$$P(X(t_1 + h) \in B_1, ..., X(t_n + h) \in B_n) = P(X(t_1) \in B_1, ..., X(t_n) \in B_n)$$

If $E[|X(t)|] < \infty$, then from the above equation it follows that

$$E[X(t)] = E[X(t + h)] = m \text{ a constant}$$

Similarly, if the second moment of the process exists, then it is also equal to a constant. We set $T = \mathbb{R}^+$ and let $s, t \in T$, and consider the covariance

$$R(s, t) = E[X(s)X(t)]$$

Clearly, $R(s, t) = R(t, s)$. If we assume that $s < t$, then we get

$$E[X(s)X(t)] = E[X(s + h)X(t + h)]$$

When we set $h = -s$ in the above equation, we get

$$R(s, t) = R(0, t - s)$$

Definition 25.1.24 *A real or complex random variable Z on $(\Omega, \mathcal{B}, P)$ is said to be second order if $E[|Z|^2] < \infty$. The family of all second-order r.v.'s on $(\Omega, \mathcal{B}, P)$ is denoted by*

$$L_2 = L_2(\Omega, \mathcal{B}, P)$$

Definition 25.1.25 *A stochastic process $(X(t); t \in T)$ on $(\Omega, \mathcal{B}, P)$ is second order if*

$$X(t) \in= L_2(\Omega, \mathcal{B}, P)\,,\ \forall t \in T$$

4. Wide sense stationary processes. It is a class of stochastic processes whose solution requires only the knowledge of the first two moments of a process. In that case, problems are considerably simplified if these moments are invariant under time-shift.

   Definition 25.1.26 *A second-order stochastic process $(X(t); t \in T)$ on a probability space $(\Omega, \mathcal{B}, P)$ is said to be wide sense stationary if, for all $t \in T$,*

   $$E[X(t)] = m \text{ (constant) and } R(t, t + \tau) = R(0, \tau)$$

   Thus, the second-order strictly stationary stochastic process is a fortiori wide sense stationary.

5. Martingales. Let $(X(t); t \in T)$ be a real stochastic process on $(\Omega, \mathcal{B}, P)$ such that $E[|X(t)|] < \infty$ for all $t \in T$.

   Definition 25.1.27 *Martingale*
   The stochastic process $X(t)$ is said to be a martingale if, for every $t_1 < .... < t_n \in T$,

   $$E[X(t_n)|X(t_1), ..., X(t_{n-1})] = X(t_{n-1}) \text{ a.s.}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

If

$$E[X(t_n)|X(t_1),...,X(t_{n-1})] \leqslant X(t_{n-1}) \text{ a.s.}$$

the process $X(t)$ is a super-martingale. Finally, if

$$E[X(t_n)|X(t_1),...,X(t_{n-1})] \geqslant X(t_{n-1}) \text{ a.s.}$$

the process $X(t)$ is a sub-martingale.

We are now going to define three types of continuity of a stochastic process. The first one is stochastic continuity (or continuity in probability). Let $\{X(t); t \in T\}$ be a real-valued stochastic process on $(\Omega, \mathcal{B}, P)$.

Definition 25.1.28 *Stochastic continuity*
The process $\{X(t); t \in T\}$ is said to be stochastically continuous at a point $t_0 \in T$, if for any $\epsilon > 0$,

$$\lim_{t \to t_0} P(|X(t) - X(t_0)| > \epsilon) = 0 \tag{25.1.7}$$

If that equation holds in every point $t_0 \in T$, the process is called stochastically continuous on T. From this definition, we see that stochastic continuity is a regularity condition on bivariate marginal distributions of the process. We have

$$P(|X(t) - X(t_0)| > \epsilon) = 1 - \int_{-\infty}^{\infty} P(X(t_0) \in dx) \int_{x-\epsilon}^{x+\epsilon} P(X(t) \in dy | x(t_0) = x)$$

As an example, we let $\{T_k\}_1^\infty$ be an i.i.d. sequence of non-negative r.v.'s on a probability space $(\Omega, \mathcal{B}, P)$ with $H(t) = P(T_i \leqslant t)$ continuous. Set

$$X(t) = \sum_{i=1}^{n} I_{T_i \leqslant t}, t \geqslant 0$$

Clearly, $0 \leqslant X(t_1) \leqslant X(t_2)$ for all $0 \leqslant t_1 \leqslant t_2$. That is, the realisations of $X(t)$ are non-decreasing step functions with unit jumps at points

$$T_{n1} < T_{n2} < ... < T_{nn}$$

where $\{T_{ni}\}_1^n$ are order statistics for $\{T_i\}_1^n$. Although every sample function of $X(t)$ has discontinuities, the random process $X(t)$ is stochastically continuous on $[0, \infty)$ since due to Markov's inequality and continuity of $H(\cdot)$, it follows that

$$P(|X(t \pm h) - X(t)| > \epsilon) \leqslant \frac{n}{\epsilon} |H(t \pm h) - H(t)| \to 0$$

as $h \to 0^+$ for each $t \geqslant 0$. The condition in Equation (25.1.7) is equivalent to

$$X(t) \xrightarrow{P} X(t_0) \text{ as } t \to t_0$$

Thus, a process can be stochastically continuous even if each realisation of the process is discontinuous. This is so when the probability that a discontinuity will occur in an instant $t \in T$ is zero. On the other hand, a process is stochastically discontinuous at a point $t_0 \in T$ only if t_0 is a fixed discontinuity point of the process.

Definition 25.1.29 *Suppose that $\{X(t); t \in T\}$ is real-valued and separable, and denote by N_{t_0} the set of sample functions which are discontinuous at $t_0 \in T$. If $P(N_{t_0}) > 0$, we say that $t_0 \in T$ is a fixed discontinuity point of $X(t)$.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 25.1.30 *Suppose that $\{X(t); t \in T\}$ is real-valued and separable. The stochastic process is continuous (a.s.) at a point $t_0 \in T$ if the set of realisations discontinuous at t_0 is negligible. If the process is (a.s.) continuous at every $t \in T$, we say that the process is (a.s.) continuous on T.*

We see that every separable process continuous (a.s.) at point t is also continuous in probability at t (in this case $P(N_t) = 0$), but the converse is false, in general.

Definition 25.1.31 *A stochastic process $\{X(t); t \in T\}$ is said to have (a.s.) continuous trajectories if the set of those trajectories which have discontinuities on T is negligible.*

If $\{X(t); t \in T\}$ is (a.s.) continuous at each $t \in T$, it does not necessarily imply that is has (a.s.) continuous trajectories. The set of all those trajectories of the process without discontinuities on T is

$$N^c = \bigcap_{t \in T} N_t^c$$

and this event may not have probability 1.
The following theorem by Kolmogorov gives a sufficient condition for (a.s.) continuity of sample functions of a separable stochastic process.

***Proposition* 38** *Let $\{X(t); t \in T\}$ be a real-valued separable process and T a compact interval. If there exist constants $\alpha > 0$, $\beta > 0$, $C > 0$ such that*

$$E[|X(t+h) - X(t)|^\alpha] \leqslant Ch^{1+\beta}$$

then almost all sample functions $X(t)$ are continuous on T.

Going back to the concept of separability introduced by Doob, for a given real-valued process, there is several equivalent versions, some of which are separable. Thus, we should choose a particular separable version with sample paths having nice properties.
We consider the stochastic process $\{X(t); t \in T\}$ without discontinuity points of the second kind, that is,

$$X(t-0, \omega) \text{ and } X(t+0, \omega)$$

exist for all $t \in T$ and $\omega \in \Omega$. In such case, the following proposition states that we can always choose a version $\tilde{X}(t)$ of $X(t)$ which is continuous from the right if $X(t)$ is stochastically continuous.

***Proposition* 39** *Let $\{X(t); t \in T\}$ be real-valued stochastically continuous random process on a complete probability space $(\Omega, \mathcal{B}, P)$ and $T \subset \mathbb{R}$ a compact interval. If the process does not have discontinuities of the second kind, there exists a separable version $\tilde{X}(t)$ equivalent to $X(t)$ whose sample functions are continuous from the right.*

Before discussing measurable random process we need to introduce the following continuity proposition:

***Proposition* 40** *Continuity in probability*
Let $\{X(t); t \in T\}$ be a real-valued stochastic process and $T \subset \mathbb{R}$ a compact interval. A necessary and sufficient condition for $X(t)$ to be continuous in probability on T is that

$$\sup_{|t-s|<h} P(|X(t) - X(s)| > \epsilon) \to 0$$

as $h \to 0$ for any $\epsilon > 0$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We let $\mathcal{R}$ be the σ-algebra of Borel subsets of $\mathbb{R}$, and use the notation

$$\mathcal{R}_C = \mathcal{R} \cap C = \{B \cap C; B \in \mathcal{R}\}$$

where $\mathcal{R}_C$ is a σ-algebra such that $\mathcal{R}_C \subset \mathcal{R}$ if $C \in \mathcal{R}$. From Definition (25.1.11), $X(t,\omega)$ is a $\mathcal{B}$-measurable function on Ω for every $t \in T$. However, it does not follow that the mapping

$$X(\cdot,\cdot) : T \times \Omega \to \mathbb{R}$$

is $\mathcal{R}_T \times \mathcal{B}$-measurable.

Definition 25.1.32 *A real-valued stochastic process $\{X(t); t \in T\}$ is said to be measurable if the mapping above is $\mathcal{R}_T \times \mathcal{B}$-measurable, that is, if $\{(\omega,t); X(t,\omega) \in B\} \in \mathcal{R}_T \times \mathcal{B}$ for every $B \in \mathcal{R}$.*

If $\{X(t); t \in T\}$ is a measurable process, then from Fubini's theorem (see Theorem (**??**)), it follows that for almost all $\omega \in \Omega$ the sample functions $X(\cdot,\omega)$ are $\mathcal{R}_T$-measurable.
To understand the concept of measurability, suppose that for a measurable process $X(t)$, we have

$$\int\int_{T\times\Omega} |X(t,\omega)| dt P(d\omega) < \infty$$

Again, invoking Fubini's theorem, we get

$$\int\int_{T\times\Omega} |X(t,\omega)| dt P(d\omega) = \int_T E[|X(t)|]dt = \int_\Omega P(d\omega) \int_T |X(t,\omega)| dt$$

Thus,

$$\int_T E[|X(t)|]dt < \infty$$

and

$$Z(\omega) = \int_T |X(t,\omega)| dt < \infty \text{ (a.s.)}$$

where $Z(\cdot)$ is a r.v. on $(\Omega, \mathcal{B}, P)$.
We now give some sufficient conditions ensuring measurability of a random process.

***Proposition* 41** *Let $\{X(t); t \in T\}$ be a real-valued stochastic process and $T \subset \mathbb{R}$ a compact interval. If $X(t)$ is continuous in probability, there exists a version $X(t)$ which is separable and measurable.*

25.2 Introduction to Markov processes

25.2.1 Defining Markov processes

Let $\{\xi(t); t \in T\}$ be a Markov process (see Definition (25.1.19)) on a probability space $(\Omega, \mathcal{B}, P)$ and state space $\{S, \mathscr{L}\}$ of a process where S is the set of values of the process and $\mathscr{L}$ is a σ-algebra of subsets of S. We let $\pi(\cdot)$ be a probability distribution on the state space $\{S, \mathscr{L}\}$.
We consider homogeneous Markov process, as in Definition (25.1.22), with transition probability $P(x,t,B)$, as in Definition (25.1.21), and $\pi(\cdot)$ as its initial distribution (see Definition (25.1.20)). The process $\xi(t)$ is said to be a homogeneous Markov process if its transition probability has the property

$$P(\xi(s+t) \in B | \xi(s) = x) = P(x,t,B) \tag{25.2.8}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

for all $t > 0$, $s + t \in T$, $x \in S$ and $B \in \mathscr{L}$.
Note, we read the transition probability $P(x, t, B)$ from left to right.
We denote P_π the probability measure on the measurable space $\{S^{\mathbb{R}^+}, \mathscr{L}^{\mathbb{R}^+}\}$, where $\mathbb{R}^+ = [0, \infty)$. For any $x \in S$, we write

$$P_x(\cdot) = P_\pi(\cdot|\xi(0) = x) \tag{25.2.9}$$

Further, since

$$\pi(B) = P_\pi(\xi(0) \in B)$$

we have

$$P_\pi(G) = \int_S P_x(G)\pi(dx)$$

where $G \in \mathscr{L}^{\mathbb{R}^+}$. We denote E_x the expectation operator corresponding to P_x.
Given Dynkin's definition of a Markov process (see Dynkin [1965]), its dynamics exist up to a certain random time τ and then terminates. Thus, we only consider Markov processes for which $P(\tau(\omega) = +\infty) = 1$, where τ is the random time where $\xi(t, \omega)$ vanishes. Such processes are called conservative.
Further, we assume that no transaction can take place in zero time. That is,

$$P(x, 0, B) = \begin{cases} 1 \text{ if } x \in B \\ 0 \text{ if } x \notin B \end{cases}$$

and we also assume that $P(x, t, B)$ is continuous at zero, that is,

$$\lim_{t \to 0^+} P(x, t, B) = \begin{cases} 1 \text{ if } x \in B \\ 0 \text{ if } x \notin B \end{cases}$$

for all $x \in S$ and $B \in \mathscr{L}$.
If $\xi(t)$ is a (not necessarily homogeneous) Markov process with transition probability $P(s, t, x, B)$, $0 \leqslant s < t$, one can show that if $\xi(t)$ is strictly stationary, then it must be a homogeneous Markov process. It follows from

$$\begin{aligned} P(s, t, x, B) &= P(\xi(t) \in B|\xi(s) = x) \\ &= P(\xi(t - s) \in B|\xi(0) = x) = P(0, t - s, x, B) = P(x, t - s, B) \end{aligned}$$

However, in general, the converse does not hold. To check whether or not a homogeneous Markov process is strictly stationary, we need the concept of an invariant measure.

Definition 25.2.1 *Invariant measure*
An invariant measure of a homogeneous Markov process with state space $\{S, \mathscr{L}\}$ and transition probability $P(x, t, B)$ is a measure μ on $\{S, \mathscr{L}\}$ satisfying the following condition: for each $B \in \mathscr{L}$ and $t \geqslant 0$

$$\mu(B) = \int_S P(x, t, B)\mu(dx)$$

Note, the invariant measure is not necessarily a probability measure on $\{S, \mathscr{L}\}$. When $\mu(S) = 1$, it is called a stationary distribution of the Markov process.
If $\{\xi(t); t \geqslant 0\}$ is a strictly stationary Markov process, then it is homogeneous, and for all t,

$$\pi(B) = P(\xi(t) \in B)\,, B \in \mathscr{L}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

From this we obtain

$$\pi(B) = \int_S P(x,t,B)\pi(dx)$$

Consequently, every one-dimensional marginal distribution of the process is an invariant measure.
Conversely, if μ is a stationary distribution of a homogeneous Markov process, then the process is also strictly stationary under P_π.
In fact, since μ can be considered as the initial distribution of $\xi(t)$, we have that for any $t > 0$ and $B \in \mathscr{L}$,

$$\begin{aligned} P(\xi(t) \in B) &= \int_S P(\xi(0) \in dx, \xi(t) \in B) \\ &= \int_S P(x,t,B)\mu(dx) = \mu(B) \end{aligned}$$

From this, we deduce that

$$\begin{aligned} P(\xi(t_1+s) \in B_1, \xi(t_2+s) \in B_2) &= \int_S \int_{B_1} \int_{B_2} P(\xi(s) \in dx, \xi(t_1+s) \in dx_1, \xi(t_2+s) \in dx_2) \\ &= \int_S \int_{B_1} \int_{B_2} P(x,t_1,dx_1)P(x_1,t_2-t_1,dx_2)\mu(dx) \\ &= \int_S P(\xi(0) \in dx, \xi(t_1) \in B_1, \xi(t_2) \in B_2) \\ &= P(\xi(t_1) \in B_1, \xi(t_2) \in B_2) \end{aligned}$$

and so on.

25.2.1.0.1 Strong Markov processes In order to introduce strong Markov processes, we first need to define the concept of stopping time of a process. So, we consider the stochastic process $\{X(t); t \in \mathbb{R}^+\}$ defined on a probability space $(\Omega, \mathcal{B}, P)$ with an arbitrary state space $\{S, \mathscr{L}\}$. We assume that the process is directly given, that is, $\Omega = S^{\mathbb{R}^+}$ and $\mathscr{L} = \mathscr{L}^{\mathbb{R}^+}$, so that each $\omega \in \Omega$ is a function $\omega : \mathbb{R}^+ \to S$.

Definition 25.2.2 *A random variable*

$$\tau : \Omega \to [0, \infty]$$

is called a stopping time if it has the property that for any two $\omega_1, \omega_2 \in \Omega$ such that

$$\omega_1(s) = \omega_2(s)\,,\ \forall 0 \leqslant s < t$$

and $\tau(\omega_1) \leqslant t$, then $\tau(\omega_1) = \tau(\omega_2)$.

As a consequence of the above definition, the occurrence or non-occurrence of the random event $\{\omega; \tau \leqslant t\}$ can be made from the observation of the random process $X(s)$ on $[0,t]$ alone.
We now consider a homogeneous Markov process $\{\xi(t); t \geqslant 0\}$ with an arbitrary state space $\{S, \mathscr{L}\}$. These properties of $\xi(t)$ imply that, for any $0 \leqslant s_1 < ... < s_m < s$ and $0 < t_1 ... < t_n$, (following notation in Equation (25.2.9))

$$P(\xi(s+t_1) \in B_1, ..., \xi(s+t_n) \in B_n | \xi(s_1) = x_1, ..., \xi(s_m) = x_m, \xi(s) = x) = P_x(\xi(t_1) \in B_1, ..., \xi(t_n) \in B_n)$$

where $\{B_i\}_1^n \subset \mathscr{L}$. In general, we can not expect the above relation to hold if s is replaced with a stopping time τ. Markov processes for which this is true are called strong Markov processes.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 25.2.3 *A measurable homogeneous Markov process with a state space S is said to be have the strong Markov property if, for any stopping time τ,*

$$P(\xi(\tau+t_1)\in B_1,...,\xi(\tau+t_n)\in B_n|\xi(s);s<\tau\,,\,\xi(\tau)=x)=P_x(\xi(t_1)\in B_1,...,\xi(t_n)\in B_n)$$

The following result, due to Dynkin and Yushkevich, discuss when a homogeneous Markov process has the strong Markov property (see Dynkin et al. [1956] [1979]).

***Proposition* 42** *A homogeneous Markov process $\{\xi(t);t\geqslant 0\}$ with state space $\{S,\mathscr{L}\}$ and transition probability $P(x,t,B)$ has the strong Markov property if and only if for any $x\in S$, $B\in\mathscr{L}$, and $t>0$,*

$$P_x(\xi(\tau+t)\in B)=\int_S P(y,t,B)P_x(\xi(\tau)\in dy)$$

Suppose that S is a metric space, then $\mathscr{L}$ is the Borel algebra. We let **B** be the space of all bounded Borel measurable functions on S. With the usual supremum norm

$$\|h\|=\sup_x|h(x)|$$

the space **B** is a Banach space. For each $t\geqslant 0$, define an operator T^t by

$$(T^th)(x)=\int_S h(y)P(x,t,dy) \tag{25.2.10}$$

or equivalently

$$(T^th)(x)=E_x[h(\xi(t))] \tag{25.2.11}$$

Since

$$|(T^th)(x)|=\Big|\int_S h(y)P(x,t,dy)\Big|\leqslant\sup_y|h(y)|=\|h\|$$

$T^t:\mathbf{B}\to\mathbf{B}$ is a contraction.

***Proposition* 43** *Let $\{\xi(t);t\geqslant 0\}$ be a homogeneous Markov process with state space $\{S,\mathscr{L}\}$ and a transition probability $P(x,t,B)$ which has the property that the operation in Equation (25.2.10) maps continuous bounded functions into continuous bounded functions. Then, the process $\xi(t)$ possesses the strong Markov property.*

Processes satisfying conditions of Proposition (43) are called Feller processes. Yushkevich showed that, given a homogeneous Markov process $\{\xi(t);t\geqslant 0\}$, if it is stochastically continuous, there exists a separable version which has the strong Markov property. That is, if all elements of S are stable states, the Markov chain is a strong Markov process.

25.2.2 Higher-dimensional Markov chains

We first discuss the structure of a 2D Markov chain and introduce high-dimensional Markov chain for categorical data sequences.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

25.2.2.1 2D Markov chains

We briefly introduce 2D Markov chains with time support given by the discrete plane $\mathbb{Z} \times \mathbb{Z}$, and discuss the separation property characterising their dynamics. We follow Fornasini [1990].
We consider any finite homogeneous Markov chain with n states $s_1, s_2, ..., s_n$. The transitions from one state to another occur at times $..., 0, 1, ...$, and the probabilistic picture of possible changes at each step is given by a stochastic matrix $A \in \mathbb{R}_+^{n \times n}$. Once we know the probabilities

$$X_t = \left[x_1(t), x_2(t), ..., x_n(t)\right]$$

of the various states at time t, the probabilities after one step are the components of the raw vector

$$X_{t+1} = X_t A \tag{25.2.12}$$

which is a discrete-time, affine dynamical system (see Equation (23.4.11).
The separation information states that knowing the outcome of the last experiment, we can neglect any other information we have about the past in predicting the future. Hence, Markov chains are a special subclass of positive linear dynamical systems (see Appendix (23.4)) evolving autonomously on the set

$$\mathcal{X} = \{X \in \mathbb{R}_+^n \mid \sum_{h=1}^{n} x_h = 1\}$$

of n-dimensional probability vector.
We consider dynamical patterns evolving in the discrete plane $\mathbb{Z} \times \mathbb{Z}$, partially ordered by the product of the orderings

$$(r, s) \leqslant (h, k) \text{ iff } r \leqslant h \text{ and } s \leqslant k$$

These constraints imply that the configuration attained at (k, k) only depends on the configurations and input values at $(r, s) \leqslant (h, k)$. This autonomous 2D system constitutes the easiest non-trivial instance of these dynamical behaviours. In that setting, the local configuration $X_{h+1,k+1}$ is linearly determined by the nearest past configurations $X_{h,k+1}$ and $X_{h+1,k}$. As a result, we get the following first-order updating equation:

$$X_{h+1,k+1} = X_{h,k+1}A^{(1)} + X_{h+1,k}A^{(2)} \tag{25.2.13}$$

where X is an n-dimensional real-valued vector, and $A^{(i)}$, $i = 1, 2$, are $n \times n$ real matrices. Here, the separation property applies in this 2D environment. The local configuration $X_{h+1,k+1}$ does not require any information about the system history in the past cone $\{(r, s) < (h + 1, k + 1)\}$, with the exception of the nearest points $(h, k + 1)$ and $(h + 1, k)$. We obtain a 2D theory of Markov chains by introducing suitable constraints in Equation (25.2.13). They must guarantee that any pair of probability vectors $X_{h,k+1}$ and $X_{h+1,k}$ leads in turn to a new probability vector at $X_{h+1,k+1}$, so that the components of $X_{h+1,k+1}$ can be viewed as probabilities of the various states at point $(h+1, k+1)$.
The author considered a system-theoretic description of 2D Markov chains and an outline of some of their internal properties.

1. The first property relates to the algebraic structure of the matrices $A^{(i)}$, $i = 1, 2$, of transition probabilities. It states that the pair $(A^{(1)}, A^{(2)})$ can be written as

$$(aP, (1 - a)Q)$$

where P and Q are stochastic matrices and $0 \leqslant a \leqslant 1$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

2. The second property is a remarkable restriction on the variety $\mathcal{V}(\Delta)$ of the characteristic 2D polynomial

$$\Delta(z_1, z_2) = det(I - A^{(1)}z_1 - A^{(2)}z_2)$$

The author showed that $\mathcal{V}(\Delta)$ intersects the unit closed polydisk $\mathcal{P}_1$ only at some points of its distinguished boundary $\mathcal{F}_1$.

3. The third result established a connection between the intersection $\mathcal{V}(\Delta) \cap \mathcal{F}_1$ and the long-term behaviour of the probability vectors $X(h, k)$.

The structure of a 2D Markov chain $\mathcal{M}$ with n states $s_1, s_2, ..., s_n$ is given by

1. an autonomous 2D system

$$X(h+1, k+1) = X(h, k+1)A^{(1)} + X(h+1, k)A^{(2)}$$

of dimension n, with the property that $X(h+1, k+1)$ is a probability row vector for every pair of probability row vectors $X(h, k+1)$ and $X(h+1, k)$.

2. a sequence of initial probability vectors

$$\mathcal{X}_0 = \{X(h, k) \mid (k, k) \in \mathcal{C}_0 \, , X(h, k) \in X\}$$

where

$$\mathcal{C}_0 = \{(h, k) \in \mathbb{Z} \times \mathbb{Z} \mid h + k = 0\}$$

is a separation set in $\mathbb{Z} \times \mathbb{Z}$, and $x_i(h, k)$, $i = 1, 2, ..., n$, denotes the probability that s_i is the state of the system at the initial point (h, k).

The pair $(A^{(1)}, A^{(2)})$ determines the probabilistic behaviour of the system, once the probability distributions are given at every point of $\mathcal{C}_0$. Given Equation (25.2.13), we now want to find the circumstances under which we can be certain that the new vector $X_{h+1,k+1}$ will also be of the same type. One partial answer is the following:

Lemma 25.2.1 *Let P and Q be $n \times n$ stochastic matrices, and consider any real number a in the interval $[0, 1]$. Then $A^{(1)} = aP$ and $A^{(2)} = (1 - a)Q$ are matrices of a 2D Markov chain.*

In order to make this Lemma canonical for 2D Markov chains, we need a preliminary result on the uniqueness of Equation (25.2.13). Giving an n-state 2D Markov chain $\mathcal{M}$ essentially reduces to assigning a one-step transition probability map

$$\pi : X \times X \to X$$

via the restriction to $X \times X$ of a suitable linear map from $\mathbb{R}^n \times \mathbb{R}^n$ into $\mathbb{R}^n$. We want to know if the linear map producing π is uniquely determined by π, that is, is the pair $(A^{(1)}, A^{(2)})$ that realises π unique. It is answered in the following lemma.

Lemma 25.2.2 *Assume $(A^{(1)}, A^{(2)})$ is a pair of $n \times n$ matrices of a 2D Markov chain $\mathcal{M}$ with n states that realises the transition map π. Then, for any matrix M with all rows the same vector, the pair $(A^{(1)} + M, A^{(2)} - M)$ realises the same transition map. Vice versa, if $(A^{(1)}, A^{(2)})$ and $(\overline{A}^{(1)}, \overline{A}^{(2)})$ realise π, then there exists a matrix M with all rows the same vector such that*

$$\overline{A}^{(1)} = A^{(1)} + M \, , \, \overline{A}^{(2)} = A^{(2)} - M$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

As a result, given a 2D Markov chain with n states, there are infinitely many chains equivalent to it (realising the same probability transition map). We can now prove that each equivalence class includes at least one 2D Markov chain represented by a convex combination of two stochastic matrices.

Theorem 25.2.1 *A 2D Markov chain with n states can be represented as*

$$X_{h+1,k+1} = X_{h,k+1}aP + X_{h+1,k}(1-a)Q$$

where P and Q are $n \times n$ stochastic matrices and $0 \leqslant a \leqslant 1$.

This theorem completely clarifies the class of dynamical models described in Equation (25.2.13). It defines a canonical 2D Markov chain denoted by $\mathcal{M} = (a, P, Q)$. We can characterise the pairs $(A^{(1)}, A^{(2)})$ that provide 2D Markov chains equivalent to the following ones

$$X_{h+1,k+1} = X_{h+1,k}\overline{A}^{(2)}$$

or

$$X_{h+1,k+1} = X_{h,k+1}\overline{A}^{(1)}$$

An equivalent 2D Markov chain has matrices

$$A^{(1)} = M \text{ , } A^{(2)} = \overline{A}^{(2)} - M$$

or

$$A^{(1)} = \overline{A}^{(1)} - M \text{ , } A^{(2)} = M$$

where M is an arbitrary $n \times n$ matrix with all rows the same vector. The converse is also true.

25.2.2.2 Categorical data

We are briefly going to introduce high-dimensional Markov chain for categorical data sequences. We follow Ching et al. [2004].

25.2.2.2.1 First-order Markov chain We let $\{X_t\}$ be the observed time series data sequence with m states, we choose to count the transition frequency F_{jk} in the observed sequence from state k to state j in one step. Thus, we can construct the one-step transition frequency matrix for the observed sequence as follows:

$$F = \begin{pmatrix} F_{11} & \cdots & \cdots & F_{1m} \\ F_{21} & \cdots & \cdots & F_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ F_{m1} & \cdots & \cdots & F_{mm} \end{pmatrix}$$

Knowing F, we can get the estimates for P_{kj} (column normalisation) as follows:

$$P = \begin{pmatrix} P_{11} & \cdots & \cdots & P_{1m} \\ P_{21} & \cdots & \cdots & P_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ P_{m1} & \cdots & \cdots & P_{mm} \end{pmatrix}$$

where

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$P_{kj} = \frac{F_{kj}}{\sum_{r=1}^{m} F_{rj}}$$

is the maximum likelihood estimator.
As an example, we consider the categorical data sequence of three states

$$\{1, 1, 2, 2, 1, 3, 2, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 1, 2\}$$

We let the sequence $\{X_t\}$ be represented in vector form:

$$X_0 = (1, 0, 0)^\top, X_1 = (1, 0, 0)^\top, X_2 = (0, 1, 0)^\top, ..., X_{19} = (0, 1, 0)^\top$$

The first-order Markov chain model can be described as follows: We count the transition frequency from state k to state j in the sequence and construct the transition frequency matrix F and the transition probability matrix $\widehat{Q}$ for the sequence

$$F = \begin{pmatrix} 1 & 3 & 3 \\ 6 & 1 & 1 \\ 1 & 3 & 0 \end{pmatrix} and\ \widehat{Q} = \begin{pmatrix} \frac{1}{8} & \frac{3}{7} & \frac{3}{4} \\ \frac{6}{8} & \frac{1}{7} & \frac{1}{4} \\ \frac{1}{8} & \frac{3}{7} & 0 \end{pmatrix}$$

and the first-order Markov chain model is

$$X_{t+1} = \widehat{Q} X_t$$

We now introduce a few proposition related to steady-state probability distribution.

***Proposition* 44** *Given an irreducible and aperiodic Markov chain of m states, then for any initial probability distribution X_0*

$$\lim_{t \to \infty} \|X_t - \pi\| = \lim_{t \to \infty} \|P^t X_0 - \pi\| = 0$$

where π is the steady-state distribution of the transition probability matrix P of the underlying Markov chain and $\pi = P\pi$.

The Perron-Frobenius theorem is given below:

***Proposition* 45** *Let A be a non-negative and irreducible square matrix of size m. Then*

- *A has a positive real eigenvalue λ equal to its spectral radius*

$$\lambda = \max_{1 \leqslant k \leqslant m} |\lambda_k(A)|$$

  where $\lambda_k(A)$ denotes the kth eigenvalue of A.

- *There corresponds an eigenvector z, its entries being real and positive, such that $Az = \lambda z$.*
- *The eigenvalue λ is a simple eigenvalue of A.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

25.2.2.2.2 Higher-order Markov chain Higher-order Markov chain can better model categorical data sequence to capture the delay effect. A conventional nth order Markov chain of m states has $O(m^n)$ states and therefore parameters. In that case, the number of transition probabilities to be estimated increases exponentially with respect to the order n of the model, leading to computational problems.
Raftery [1985] proposed a high-order Markov chain model involving only one additional parameter for each extra lag. The model is given by

$$P\big(X_t = k|\ X_{t-1} = k_1, ..., X_{t-n} = k_n\big) = \sum_{i=1}^{n} \lambda_i q_{kk_i}$$

where $k, \{k_i\}_{i=1}^n \in M$ and $M = \{1, 2, ..., m\}$ is the set of the possible states. Further, we have

$$\sum_{i=1}^{n} \lambda_i = 1 \text{ and } Q = [q_{ij}]$$

where Q is a transition probability matrix with column sums equal to one, such that

$$0 \leqslant \sum_{i=1}^{n} \lambda_i q_k k_i \leqslant 1\ ,\ k, k_1, ..., k_n = 1, 2, ..., m$$

He proved that this model was analogous to the standard $AR(n)$ model in time series. The parameters $q_{k_0 k_i}, \lambda_i$ can be estimated numerically by maximising the log-likelihood of the above model subject to the above constraints. However, this approach involves solving a highly non-linear optimisation problem. Further, the proposed numerical method neither guarantees convergence nor a global maximum.
Ching [2004] generalised that model as follows:

$$X_{t+n+1} = \sum_{i=1}^{n} \lambda_i Q_i X_{t+n+1-i}$$

where Q_i is the ith step transition probability matrix of the sequence. It is assumed that the λ_i are non-negative such that $\sum_{i=1}^n \lambda_i = 1$. Hence, the RHS of the above model is a probability distribution.
We now present some of the properties of the model.

***Proposition* 46** *If Q_i is irreducible, $\lambda_i > 0$ for $i = 1, 2, ..., n$ and $\sum_{i=1}^n \lambda_i = 1$, then the above model has a stationary distribution $\overline{X}$ when $t \to \infty$ independent of the state vectors $X_0, X_1, ..., X_{n-1}$.*

This proposition gives a sufficient condition for the sequence X_t to converge to a stationary distribution The stationary distribution $\overline{X}$ is the unique solution to the linear system of equations

$$\big(I - \sum_{i=1}^{n} \lambda_i Q_i\big)\overline{X} = 0 \text{ and } 1^\top \overline{X} = 1$$

where I it the $m \times m$ identity matrix.
We can estimate the ith step transition probability matrix Q_i by counting the transition frequency $f_{jk}^{(i)}$ in the sequence from state k to state j in the ith step. We get

$$F^{(i)} = \begin{pmatrix} f_{11}^{(i)} & \cdots & \cdots & f_{m1}^{(i)} \\ f_{12}^{(i)} & \cdots & \cdots & f_{m2}^{(i)} \\ \vdots & \vdots & \vdots & \vdots \\ f_{1m}^{(i)} & \cdots & \cdots & f_{mm}^{(i)} \end{pmatrix}$$

for $i = 1, 2, ..., n$. From $F^{(i)}$, we get by column normalisation

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\widehat{Q}^{(i)} = \begin{pmatrix} \widehat{q}_{11}^{(i)} & \cdots & \cdots & \widehat{q}_{m1}^{(i)} \\ \widehat{q}_{12}^{(i)} & \cdots & \cdots & \widehat{q}_{m2}^{(i)} \\ \vdots & \vdots & \vdots & \vdots \\ \widehat{q}_{1m}^{(i)} & \cdots & \cdots & \widehat{q}_{mm}^{(i)} \end{pmatrix}$$

where

$$q_{kj}^{(i)} = \frac{f_{kj}^{(i)}}{\sum_{r=1}^{m} f_{rj}^{(i)}}$$

The stationary distribution $\overline{X}$ can be estimated from the sequence $\{X_t\}$ by computing the proportion of the occurrence of each state in the sequence. It is denoted by $\widehat{X}$. Then, one would expect

$$\sum_{i=1}^{n} \lambda_i \widehat{Q}_i \widehat{X} \approx \widehat{X}$$

As a result, one way of estimating the parameters is to let

$$\lambda = \big(\lambda_1, ..., \lambda_n\big)$$

and consider the following optimisation problem

$$\min_{\lambda} \| \sum_{i=1}^{n} \lambda_i \widehat{Q}_i \widehat{X} - \widehat{X} \|_{\infty} = \min_{\lambda} \max_{k} |\big[\sum_{i=1}^{n} \lambda_i \widehat{Q}_i \widehat{X} - \widehat{X}\big]_k|$$

subject to

$$\sum_{i=1}^{n} \lambda_i = 1 \text{ and } \lambda_i \geqslant 0 \,, \forall i$$

where $[\cdot]_k$ denotes the kth entry of the vector. Note, the optimisation problem can be expressed as a linear programming problem. The formulation is

$$\min_{\lambda} w$$

subject to

$$w^{\top} \geqslant \widehat{X} - \big[\widehat{Q}_1 \widehat{X} \mid \widehat{Q}_2 \widehat{X} \mid \cdots \mid \widehat{Q}_n \widehat{X}\big] \lambda^{\top}$$

and

$$w \geqslant 0 \,, \sum_{i=1}^{n} \lambda_i = 1 \text{ and } \lambda_i \geqslant 0 \,, \forall i$$

This problem was solved with Excel by Ching et al. [2004].

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 26

Monte Carlo

We briefly describe stochastic differential equations (SDEs) and their discretisation in view of computing expectation with Monte Carlo engine. These SDEs can represent the dynamics of the state of the system or stochastic processes. We use notation from option pricing theory since the option price is a particular objective function. Note, the hedge of an option (delta) can be seen as the gradient of the objective function.

See textbooks by Rubinstein [1981], Oksendal [1998], and Glasserman [2004].

26.1 The dynamics of the state of the system

26.1.1 The stochastic differential equation

We let the state process $(X_t)_{t\geqslant 0}$ be a d-dimensional Ito process valued in the open subset D with dynamics being

$$dX_t = b(X,t)dt + \sigma(X,t)dW_X(t) \tag{26.1.1}$$

where the drift $b : D \to \mathbb{R}^d$ as well as the diffusion $\sigma^\top : D \to \mathbb{R}^{d\times p}$ are regular enough to have a unique strong solution valued in D. The value of the objective $C(t,x)$ on $]0,T]\times]0,+\infty[$ is

$$C(t,X_t) = E[e^{-\int_t^T r_s ds} h(X_T)|\mathcal{F}_t] \tag{26.1.2}$$

where h is a sufficiently smooth payoff function. Furthermore, we assume that these value functions are of class $\mathcal{C}^2$ such that for $T > t$ the law of X_T knowing $X_t = x$ has a density $\phi(T,K)$ given by

$$\phi(T,K) = E_t[\delta(X_T - K)] = \frac{1}{P(t,T)}\partial_{KK}C(t,X_t;K,T)$$

where $P(t,T) = E[e^{-\int_t^T r_s ds}]$ is a zero-coupon bond (expected reward) and $\delta(\cdot)$ is the Dirac function. The density follows the Fokker-Planck equation

$$\partial_t\phi(x,t) = -\sum_{i=1}^d \partial_{x_i}\Big(b_i(x,t)\phi(x,t)+\alpha\sum_{k=1}^d\sum_{j=1}^p \partial_{x_k}\sigma_{ij}(x,t)\sigma_{kj}(x,t)\phi(x,t)\Big)+\frac{1}{2}\sum_{i=1}^d\sum_{k=1}^d\sum_{j=1}^p \partial_{x_ix_k}\sigma_{ij}(x,t)\sigma_{kj}(x,t)\phi(x,t)$$

If the dynamics are deterministic with $\sigma = 0$, we get the Liouville equation

$$\partial_t\phi(x,t) = -\sum_{i=1}^d \partial_{x_i}b_i(x,t)\phi(x,t)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Thus, if the dynamics are known but the probability distribution of the initial conditions has finite width, the cloud of possible trajectories is described by the appropriate Liouville equation. The α term corresponds to the Stratonovich systems, while for an Ito system we get $\alpha = 0$. The important thing about the noise induced drift is that it can cause the average dynamics of a system to differ from those of its deterministic counterpart, even though the white noise itself has zero mean. If the diffusion term $\sigma(x,t)$ is independent from x both Stratonovich and Ito systems yield equivalent dynamics.

26.1.2 From Stratonovich equation to Ito equation

Continuous systems, where rapidly varying quantities with small but finite correlation times are treated as white noise, follow Stratonovich calculus. For example, we can consider the dynamics

$$\frac{dx}{dt} = b(x,t) + \sigma(x,t)\xi_0(t)$$

where $\xi_0(t)$ is the rapidly varying quantity we wish to approximate as white noise. One of the restriction on the processes is that their correlation functions must decay rapidly enough in comparison with the time-scales of the deterministic functions $b(x,t)$ and $\sigma(x,t)$. The constraints should be considered in a weaker sense. Since $\xi_0(t)$ varies more rapidly than $b(x)$ and $\sigma(x)$ it can be scaled with ϵ. One need the scaled time $\tau = \frac{t}{\epsilon^2}$ such that

$$\epsilon\xi_0(\frac{t}{\epsilon^2}) = \xi(\tau)$$

Assuming $\xi(\tau)$ to be a stationary Markov process with $E[\xi(\tau)] = 0$ and $E[\xi(\tau)\xi(0)] = C(\tau)$ we get

$$E[\xi_0(t)] = 0 \text{ and } E[\xi_0(t)\xi_0(0)] = \frac{1}{\epsilon^2}C(\frac{t}{\epsilon^2})$$

To get the white noise limit we let $\epsilon \to 0$ so that $E[\xi_0(t)\xi_0(0)] \to \delta(t)$. Since $\xi(\tau)$ is a stationary Markov process, its probability distribution function (pdf) obeys a Fokker-Planck equation in the scaled coordinates. Hence, we can let $\epsilon \to 0$ and get the Fokker-Planck equation in the Stratonovich sense

$$\partial_t\phi(x) = -\partial_x b(x)\phi(x) + \frac{1}{2}\partial_x\big(\sigma(x)\partial_x(\sigma(x)\phi(x))\big)$$

corresponding to the stochastic differential equation (SDE)

$$dx = b(x)dt + \sigma(x).dW$$

Although the limiting dynamics are obtained in the limit of vanishing ϵ, we have no power over the geophysical system we wish to describe. Therefore, we may only estimate what a realistic value of ϵ would be in that system and decide whether it is small enough for the system to be usefully described by its limiting dynamics. Stratonovich calculus results when a continuous system is subjected to the white noise limit.
There do exist physical systems where the appropriate starting point is a discrete relation

$$x(t_{i+1}) = x(t_i) + b(x(t_i))\Delta t + \sigma(x(t_i))\eta(t_i)$$

where $t_{i+1} = t_i + \Delta t$ for all i and $\eta(t_i)$ are independent Gaussian random variables with $E[\eta(t_i)] = 0$ and $Var(\eta(t_i)) = \Delta t$. If the time-scales of interest are large compared with Δt we may take the continuous limit getting the SDE

$$dx = b(x)dt + \sigma(x)dW$$

The Euler scheme is a first order Runge-Kutta method used for Ito equations. Integration from the nth time step to the $(n+1)$ time step is

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$x_{n+1} = x_n + b(x_n, t)\Delta + \sigma(x_n, t)(\Delta W)_n$$

If R is a vector of centered Gaussian random numbers with autocovariance matrix being the identity matrix I, then $(\Delta W)_n = R\sqrt{\Delta}$. The Euler scheme converges to an Ito process with $o(\Delta^{\frac{1}{2}})$ accuracy. If $\sigma = \sigma(t)$ is independent from x we may use a Euler scheme for a Stratonovich system since the moments of Ito and Stratonovich additive noise processes are the same. That is, one may generate an intermediate Ito variable y and then estimate the desired Stratonovich variable x with

$$x(t) = \big(y(t + \frac{1}{2}\Delta) + y(t - \frac{1}{2}\Delta)\big)$$

In the Heun scheme, a second order Runge-Kutta method, we generate at each time step the Wiener process increment $(\Delta W)_n$ and uptate the equation as

$$x_{n+1} = x_n + \frac{1}{2}\big(b(x_n, t_n) + \mu(x^{'}_{n+1}, t_{n+1})\big)\Delta + \frac{1}{2}\big(\sigma(x_n, t_n) + \sigma(x^{'}_{n+1}, t_{n+1})\big)(\Delta W)_n$$

with the Euler predictor

$$x^{'}_{n+1} = x_n + b(x_n, t)\Delta + \sigma(x_n, t)(\Delta W)_n$$

Unfortunately, the Heun method does not converge uniformly to the correct solution with the time step. There is a transformation from every Ito equation to a Stratonovich equation and vice versa, at least in distribution. It consists of adding the term

$$\frac{1}{2}\sum_{k=1}^{d}\sum_{j=1}^{p}\partial_{x_k}\sigma_{ij}(x, t)\sigma_{kj}(x, t)$$

to the drift term $\mu_i(x, t)$ in a Stratonovich SDE to get the corresponding Ito equation. Similarly, one can subtract that term from an Ito SDE to get the corresponding Stratonovich equation. So, one can use the higher accuracy of the Heun method for an Ito equation by adjusting the drift term accordingly. This transformation is only practical for systems where $\sigma(x, t)$ is explicitly known and fairly easy to manipulate. See for example the Milstein schemes. The question of how large to make the time step naturally arises. It has been found that dividing the smallest physical time-scale by 2^6 or 2^7 yields acceptable accuracy.

26.1.3 Discretisation of the SDE

In Section (26.1.2) we briefly discussed methods needed when we simulate trajectories of a given process. We are now going to give more details on the Euler scheme and explain the reasons why it is sufficient when applied in finance. This is needed when we want to compute, using Monte Carlo simulations, a path-dependent option or an option on multi-underlyings. We want to find a scheme approximating the solution of a stochastic differential equation (SDE). For simplicity, we consider the process $(X)_{t\geqslant 0}$ given in Equation (26.1.1) and assume stationarity. Hence, its solution becomes

$$X_t = X_0 + \int_0^t b(X_s)ds + \int_0^t \sigma(X_s)dW_s \tag{26.1.3}$$

where $(W)_{t\geqslant 0}$ is a p-dimensional Brownian motion.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

26.1.3.1 The Euler scheem

We let $h = \frac{T}{m}$ be the discretisation step of the time interval $[0, T]$ and get the exact solution

$$X_h = X_0 + \int_0^h b(X_s)ds + \int_0^h \sigma(X_s)dW_s$$

so that a natural approximation of X_h is

$$X_h \approx X_0 + b(X_0)h + \sigma(X_0)(W_h - W_0)$$

Starting with $\hat{X}_0 = X_0$ and proceeding by induction, we get

$$\hat{X}_{(p+1)h} = \hat{X}_{ph} + b(\hat{X}_{ph})h + \sigma(\hat{X}_{ph})(W_{(p+1)h} - W_{ph})$$

The weak convergence states that if both b and σ are Lipsichtz functions and f is a C^4 function with bounded derivatives up to order 4, then there exists a constant C_T such that

$$|E[f(X_T)] - E[f(\hat{X}_T)]| \leqslant \frac{C_T(f)}{m}$$

The speed of convergence in L^2 is of order $h^{\frac{1}{2}}$, while for very regular functions, the speed of convergence in law of the scheme is of order h.

26.1.3.2 Justification of the Euler scheme

Various scheems of higher order have been proposed for discretising SDEs but they are often almost impossible to implement. The simplest of these scheems is the Milstein scheme (see Milshtein [1974]) but it is hard to simulate when more than one Brownian motion is involved and the speed in law remains of order h. That is, the Milstein scheme improves the trajectorial speed of convergence as it is (almost) of order h compared to $\sqrt{(h)}$ for the Euler scheme. Nevertheless, the convergence speed in law for regular function is of order h in both cases, and it is the speed in law which is relevant to evaluate the performance of Monte Carlo algorithm for option pricing. In addition, the simulation of the Milstein scheme requires the computation at each time step of all the quantities

$$\partial\sigma_j(X_{jh})\sigma_k(X_{kh})$$

making the scheme much slower than the Euler scheme. A result due to Clark and Cameron proves that in L^2-norm, the Euler scheme is in general almost optimal among the schemes involving only the random variables W_{kh} for $k \geqslant 0$. If we accept to simulate more than the increments of the Brownian motions, one can construct schemes with arbitrary high order of convergence. However, these schemes do not have much practical interest as they require to simulate iterated integrals of the Brownian motion of high order and these integrals are known to be challenging to simulate.

26.1.4 Generating trajectories

26.1.4.1 Multivariate normals

A multivariate normal distribution $N(\mu, \Sigma)$ is specified by its mean vector μ and covariance matrix Σ. The covariance matrix may be specified implicitly by its diagonal entries σ_i^2 and correlation ρ_{ij} and becomes in matrix form $\Sigma = \sigma R \sigma^\top$ where R is the correlation matrix

$$\Sigma = \begin{bmatrix} \sigma_1 & \dots & 0 \\ \dots & \dots & \dots \\ 0 & \dots & \sigma_n \end{bmatrix} \begin{bmatrix} \rho_{11} & \dots & \rho_{1n} \\ \dots & \dots & \dots \\ \rho_{n1} & \dots & \rho_{nn} \end{bmatrix} \begin{bmatrix} \sigma_1 & \dots & 0 \\ \dots & \dots & \dots \\ 0 & \dots & \sigma_n \end{bmatrix}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

From the Linear Transformation property, if $y \sim N(0, I)$ and $X = \mu + Ay$ then $X \sim N(\mu, AA^\top)$. The problem of sampling X from the multivariate normal $N(\mu, \Sigma)$ reduces to finding a matrix A for which $AA^\top = \Sigma$. One approach is to use the Cholesky Factorisation on the covariance matrix Σ. An alternative approach is to use the Cholesky Factorisation on the correlation matrix R obtaining the new matrix $\overline{R}$ such that $\overline{R}\overline{R}^\top = R$. In that case we recover the matrix A from $A = \overline{R}\sigma$.

26.1.4.2 One-dimension

In order to perform simulation of an approximate trajectory of a Brownian motion (BM) we let h be a time discretisation step. We simulate a realisation of the family of independent Gaussian random variables with mean 0 and variance 1 noted y_k for $0 \leqslant k \leqslant m$. To obtain a realisation of W_{ph} for $1 \leqslant p \leqslant m$ we compute the recursion

$$W_{ph} = \sqrt{h} \sum_{1 \leqslant k \leqslant p} y_k \tag{26.1.4}$$

More precisely, we are now going to simulate the values $\big(W(t_1), .., W(t_n)\big)$ at the fixed set points $0 < t_1 < ... < t_n$. To do so we use the fact that Brownian motion has independent normally distributed increments. We let $y_1, .., y_n$ be independent standard normal random variables. Given $t_0 = 0$ and $W(0) = 0$, subsequent values follow from

$$W(t_{i+1}) = W(t_i) + \sqrt{t_{i+1} - t_i} y_{i+1} \text{ for } i = 0, .., n-1$$

giving the increment $W(t_{i+1}) - W(t_i) = \sqrt{t_{i+1} - t_i} y_{i+1}$. From the properties of the Brownian motion we have

$$W(t_{i+1}) = \sum_{j=0}^{i-1} W(t_{j+1}) - W(t_j) = \sum_{j=0}^{i-1} \sqrt{t_{j+1} - t_j} y_{j+1}$$

For $X \sim BM(\mu, \sigma^2)$ with time-dependent coefficients, the recursion becomes

$$X(t_{i+1}) = X(t_i) + \int_{t_i}^{t_{i+1}} \mu(s) ds + \sqrt{\int_{t_i}^{t_{i+1}} \sigma^2(s) ds} y_{i+1} \text{ for } i = 0, .., n-1$$

The joint distribution of the corresponding Brownian motion are matched at times $t_1, .., t_n$ but it says nothing when time is between t_i and t_{i+1}. Using the Euler approximation, the recursion becomes

$$X(t_{i+1}) = X(t_i) + \mu(t_i)(t_{i+1} - t_i) + \sigma(t_i)\sqrt{(t_{i+1} - t_i)} y_{i+1} \text{ for } i = 0, .., n-1$$

introducing discretisation error even at times $t_1, .., t_n$ because the increments no longer have exactly the right mean and variance. In general we need to find the mean vector and covariance matrix of $\big(W(t_1), .., W(t_n)\big)$. For a standard Brownian motion $E[W(t_i)] = 0$ and $Cov(W(s), W(t)) = s$ when $0 < s < t$ so that the covariance is $\Sigma = \min(t_i, t_j)$ and the distribution becomes $N(0, \Sigma)$.

26.1.4.3 Multi-dimensions

In multiple dimension we let $y_1, .., y_n$ be independent $N(0, I)$ random vectors in $\mathbb{R}^d$ where d is the dimension and $W(.)$ is now a standard d-dimensional Brownian motion. To simulate $X \sim BM(\mu, \Sigma)$ we find the matrix A for which $AA^\top = \Sigma$ and use the recursion

$$X(t_{i+1}) = X(t_i) + \int_{t_i}^{t_{i+1}} \mu(s) ds + A(t_i, t_{i+1}) y_{i+1} \text{ for } i = 0, .., n-1$$

with

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$A(t_i, t_{i+1})A(t_i, t_{i+1})^\top = \int_{t_i}^{t_{i+1}} \Sigma(s)ds$$

thus requiring n factorisation.

26.1.5 Construction of the Brownian bridge

26.1.5.1 Construction

We saw in Section (26.1.4.2) that one can generate the vector $(W(t_1), .., W(t_n))$ from left to right by using the recursion

$$W(t_{i+1}) = W(t_i) + \sqrt{t_{i+1} - t_i}Z_{i+1}\ , i = 0, .., n-1$$

where Z_i for $i = 1, .., n$ are independent standard normal random variables. However, we can generate the $W(t_i)$ in any order we choose as long as we sample from the correct conditional distribution at each time step given the values already generated. Conditioning a Brownian motion on its endpoints produces a Brownian bridge. Hence, we first determine $W(t_n)$ and then fill in the intermediate values by simulating a Brownian bridge from $0 = W(0)$ to $W(t_n)$. Let's assume that the Brownian path at time $s_1 < s_2 < ... < s_k$ have been determined and that we want to draw $W(s)$ conditional on these values such that $s_i < s < s_{i+1}$. From the Markov property of Brownian motion we get

$$\big(W(s)|W(s_1) = x_1, ..., W(s_k) = x_k\big) = N\big(m(s_i, s_{i+1}; s), v(s_i, s_{i+1}; s)\big)$$

with

$$m(s_i, s_{i+1}; s) = \frac{(s_{i+1} - s)x_i + (s - s_i)x_{i+1}}{s_{i+1} - s_i}\ , v(s_i, s_{i+1}; s) = \frac{(s_{i+1} - s)(s - s_i)}{s_{i+1} - s_i}$$

The conditional mean of $W(s)$ lies on the line segment connecting (s_i, x_i) and (s_{i+1}, x_{i+1}) so that $W(s)$ is normally distributed about this mean with a variance that depends on $(s - s_i)$ and $(s_{i+1} - s)$. Hence, sampling $W(s)$ from this conditional distribution, we get

$$W(s) = m(s_i, s_{i+1}; s) + \sqrt{v(s_i, s_{i+1}; s)}Z$$

with $Z \sim N(0, 1)$ independent of all $W(s_1), .., W(s_k)$. Starting by sampling $W(t_n)$ fom $N(0, t_n)$ we proceed by conditionally sampling intermediate values on the two closest time points already sampled at each step. If n is not a power of 2, one can still apply the algorithm to a subset of $2^m < n$ of the t_i with the remaining points filled in at the end. In the case where the Brownian motion has a drift μ, only sampling the rightmost point would change by sampling W_{t_m} from $N(\mu t_m, t_m)$. This is because the conditional distribution of $W(t_1), .., W(t_{n-1})$ given W_{t_m} is the same for all values of μ. Further, given a Brownian motion with a diffusion coefficient σ^2, the conditional mean is unchanged but the conditional variance is multiplied by σ^2. In the case of a d-dimensional Brownian motion we simply apply independent one-dimensional constructions to each of the coordinates. To include a drift vector it suffices to add $\mu_i t_n$ to $W_i(t_n)$ at the first step of the construction of the ith coordinate. Further, to construct $X \sim BM(\mu, \Sigma)$ we use the fact that X can be represented linearly as $X(t) = \mu t + BW(t)$ with B a (d, k) matrix for $k \leqslant d$ satisfying $BB^\top = \Sigma$ and W is a standard k-dimensional Brownian motion. We then apply a Brownian bridge construction to sample $W(t_1), .., W(t_n)$ and recover $X(t_1), .., X(t_n)$ with a linear transformation.

26.1.5.2 Justification

In general, the Brownian bridge construction has no computational advantage over the simple random walk recursion, and they both share the same distribution. However, the Brownian bridge provides us with a greater control over the coarse structure of the simulated Brownian path. It uses a single normal random variable $W(t_n) = \sqrt{t_n}y_1$ to determine

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

the endpoint of a path which may be the most important feature of the path as opposed to the combined result of n independent normal variables in the simple random walk recursion (see Equation (26.1.4)). That is, the latter proceeds by evolving the path forward through time while the former proceeds by adding increasingly fine detail to the path at each step. According to some authors such as Glasserman [2004] it is relevant to change the order in which coordinates of a sequence are assigned to arguments of an integrand when some coordinates of a low-discrepancy sequence exhibit better uniformity properties than others. This is the case for Sobol' sequences in which coordinates generated by lower-degree polynomials are preferable. That is, all coordinates of a Sobol sequence are not equally well distributed. A general strategy for improving QMC approximation is to apply a change of variables to produce an integrand for which only a small number of arguments are important and then applies the lowest-indexed coordinates of a QMC sequence to those coordinates. For instance, one can combine Sobol' sequences with a Brownian bridge. Normal procedure is for the ith coordinate of each point to be transformed to a sample from the standard normal distribution, and then scaled and summed using the random walk construction. Given that the initial coordinates of a Sobol sequence have uniformity superior to that of higher-indexed coordinates, this construction does a good job of sampling the first few increments of the Brownian path. However, many option contracts would be primarily sensitive to the terminal value of the Brownian path. On the other hand, when using the Brownian bridge construction, the first coordinate of a Sobol sequence determines the terminal value of the Brownian path, and the value becomes particularly well distributed. In addition, the first several coordinates of the Sobol sequence determine the general shape of the Brownian path, while the last few coordinates influence only the fine detail of the path. Put another way, the initial coordinates of a Sobol sequence are more uniform than the later ones such that a strategic assignment of coordinates to sources of randomness can improve accuracy. Note, neither the Brownian bridge nor the principal components construction is tailored to a particular type of option payoff and one should use additional information about the payoff to find the appropriate change of variables.

26.1.6 Simulating the state process

The state value at time T with stochastic volatility is

$$S_T = S_0 C_T e^{-\frac{1}{2}\int_0^T \sigma_s^2 ds + \int_0^T \sigma_s dW_s}$$

We are going to discretise the maturity into n time-steps such that $T = n\Delta t$ so that the state value price rewrite

$$S_T = S_0 C_T e^{-\frac{1}{2}\sum_{j=0}^{n-1}\int_{t_j}^{t_{j+1}} \sigma_j^2 ds + \sum_{j=0}^{n-1}\int_{t_j}^{t_{j+1}} \sigma_j dW_j}$$

Hence, the discretised state value at time t_i becomes

$$S_{t_i} = S_0 C_{t_i} e^{-\frac{1}{2}\sum_{j=0}^{i-1}\int_{t_j}^{t_{j+1}} \sigma_s^2 ds + \sum_{j=0}^{i-1}\sqrt{\frac{1}{\Delta t}\int_{t_j}^{t_{j+1}} \sigma_j^2} ds \hat{W}_{t_j}}$$

where $\hat{W}_{t_j} = \Delta W_j$ and $\Delta W_j = W(t_j) - W(t_{j-1})$. Recall, to compute the covariance matrix we use the matrix σ

$$\sigma = \begin{bmatrix} \sigma_1^2 & ... & 0 \\ ... & ... & ... \\ 0 & ... & \sigma_n^2 \end{bmatrix}$$

where $\sigma_i^2 = \int_{t_{i-1}}^{t_i} \sigma_s^2 ds$ for $1 \leqslant i \leqslant n$. To simulate the standard Gaussian random variable, we draw the random variable y_i from the identity matrix

$$I = \begin{bmatrix} 1 & ... & 0 \\ ... & ... & ... \\ 0 & ... & 1 \end{bmatrix}$$

so that the Brownian motion at time t_i becomes $\overline{W}_{t_i} = \sum_{j=0}^{i-1}\sqrt{\frac{1}{\Delta t}\int_{t_j}^{t_{j+1}} \sigma_j^2 ds}\hat{W}_{t_j} = \sum_{j=1}^{i}\sigma_j y_j$. In the case where the volatility is constant we get $\sigma_j = \sigma$ and the stock price re-write

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$S_{t_i} = S_0 C_{t_i} e^{-\frac{1}{2}\sigma^2(i\times\Delta t)+\overline{W}_{t_i}} = S_0 C_{t_i} \prod_{j=1}^{i} e^{-\frac{1}{2}\sigma^2\Delta t+\sigma\sqrt{\Delta t}y_j}$$

In the case where the volatility is piecewise constant, given $\hat{\sigma}_i$ in the interval $[t_{i-1}, t_i[$ we get

$$\sigma_i^2 = \int_{t_{i-1}}^{t_i} \hat{\sigma}_i^2 ds = \hat{\sigma}_i^2(t_i - t_{i-1})$$

which is the approximation made in the Euler scheme.

26.2 The Monte Carlo engine

Each replication in a Monte Carlo simulation can be interpreted as the result of applying a series of transformations to an input sequence of uniformly distributed random variables U_i with $i = 1, .., d$ producing the output $f(U_1, .., U_d)$. In control theory, f is the result of a transformation converting the U_i to normal random variables, which are transformed to paths of underlying assets, which are transformed to the discounted payoff of the objective. Given the objective value in Equation (26.1.2), the goal is to compute

$$E[f(U_1, .., U_d)] = \int_{[0,1)^d} f(x)dx$$

which is approximated by

$$\int_{[0,1)^d} f(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i) \tag{26.2.5}$$

where n is the number of simulation.

26.2.1 The setup of the Monte Carlo engine

Given a continuous process $(X_t)_{t\geqslant 0}$ defined in Appendix (26.1), we want to estimate

$$\theta = E[f(X_T)]$$

for a fixed maturity T. We take a discrete process $\hat{X}_t$ given by $\{\hat{X}_h, \hat{X}_{2h}, ..., \hat{X}_{mh}\}$ such that $mh = T$. To simplify notation we define $f_j = f(\hat{X})$ and consider the estimated value of

$$\hat{\theta} = E[f(\hat{X}_T)]$$

to be

$$\hat{\theta}_n = \frac{1}{n}\sum_{j=1}^{n} f_j$$

From the strong law of large numbers we get

$$\hat{\theta}_n \to \theta \text{ as } n \to \infty$$

Also, if we consider $Z \sim N(0,1)$ and let $Z_{1-\frac{\alpha}{2}}$ be the $(1-\frac{\alpha}{2})$ percentile point of the $N(0,1)$ distribution so that $P(-Z_{1-\frac{\alpha}{2}} \leqslant Z \leqslant Z_{1-\frac{\alpha}{2}}) = 1-\alpha$ we can recover a confidence interval. The approximate $100(1-\frac{\alpha}{2})\%$ confidence interval for θ when n is large is

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$[L(Y), U(Y)] = [\hat{\theta}_n - Z_{1-\frac{\alpha}{2}} \frac{\hat{\sigma}_n}{\sqrt{n}}, \hat{\theta}_n + Z_{1-\frac{\alpha}{2}} \frac{\hat{\sigma}_n}{\sqrt{n}}]$$

where $\hat{\sigma}_n$ is an estimated standard deviation of the process $\hat{X}_t$.

***Remark* 26.2.1** *$\hat{\theta}_n$ is an unbiased estimate of $\hat{\theta}$ but it can be a very biased estimate of θ.*

In order to measure that bias we consider the discretisation error D given by

$$D = |E[f(X_T)] - E[f(\hat{X}_T)]|$$

It leads to two types of error, the discretisation error and the statistical error.

- small value of $m \rightarrow$ greater discretisation error
- small value of $n \rightarrow$ greater statistical error

The values m and n must be chosen judiciously to control the convergence of the estimated $\hat{\theta}_n$ to the true value θ at the minimum computational cost. From the confidence interval, we see that the accuracy of the Monte Carlo pricer is governed by the relation

$$\frac{\hat{\sigma}_n}{\sqrt{n}}$$

so that we can

- multiply n by 4 to decrease the confidence interval by a factor of 2
- reduce the variance

We are now going to briefly state the different techniques that can be put in place to reduce the variance.

26.2.2 Variance reduction techniques

The situation of two square integrable r.v. X and $\tilde{X}$ such that $X \neq \tilde{X}$ having the same expectation

$$E[X] = E[\tilde{X}] = m$$

with non-zero variance $Var(X)$ and $Var(\tilde{X})$ can be reformulated by setting

$$Y = X - \tilde{X} \text{ with } E[Y] = 0 \text{ and } Var(Y) > 0$$

If $Var(X - Y) << Var(X)$ one will choose $X - Y$ to implement the Monte Carlo simulation. The aim of variance reduction techniques is to rewrite the quantity that we want to compute as the expectation of a random variable which has a smaller variance. Suppose that we want to calculate $E[X]$, then we try to re-express it as

$$E[X] = E[Y] + C$$

where Y is a random variable with lower variance than that of X and C is a known constant.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

26.2.2.1 Control variates

The idea of control variates is to write $E[f(X)]$ as

$$E[f(X)] = E[f(X) - \nu g(X)] + \nu E[g(X)] = E[\tilde{f}(X)]$$

where $\overline{g} = E[g(X)]$ can be explicitly computed and

$$var(f(X) - \nu g(X)) = var(f(X)) + \nu^2 var(g(X)) - 2\nu Cov(f, g)$$

is smaller than $var(f(X))$. This happens if f and g are strongly correlated. If we set

$$\nu^* = \frac{Cov(f, g)}{var(g)}$$

we get

$$var(\tilde{f}) = \left(1 - \rho_{f,g}^2\right) var(f)$$

which guarantees variance reduction if f and g are correlated.

26.2.2.2 Perfect control variates

In theory, a perfect control variate exist for diffusion models. Assume we want to compute $E[Z]$ where $Z = \psi(X_s, 0 \leqslant s \leqslant T)$ and X_t is the solution of

$$dX_t = b(X_t)dt + \sigma(X_t)dW_t \text{ , } X(0) = x$$

We can use the predictable representation theorem to find means of decreasing the variance of the Monte Carlo method practically.

Theorem 26.2.1 *Let Z be a random variable measurable with respect to the σ-field generated by the Brownian motion and such that $E[Z^2] < \infty$. Then there exists a stochastic process $(H_t)_{t \leqslant T}$ adapted to the Brownian filtration such that $E[\int_0^T H_s^2 ds] < \infty$ and*

$$Z = E[Z] + \int_0^T H_s dW_s$$

Even though we can cancel the variance of Z, the explicit computation of $(H_s)_{s \leqslant T}$ is much more complicated than $E[Z]$. In the special case X_t is a Markov process, under some circonstances the process (X_t) can be written as $H_t = v(t, X_t)$ a function of t and x. If we let b and σ be two Lipschitz functions, and A its infinitesimal generator of the diffusion

$$Af(x) = \frac{1}{2} \sum_{i,j=1}^{n} a_{ij}(x) \partial_{x_i x_j}^2 f(x) + \sum_{j=1}^{n} b_j(x) \partial_{x_j} f(x)$$

where $a_{ij}(x) = \sum_{k=1}^{p} \sigma_{ik}(x)\sigma_{jk}(x)$. Assume that there exists a $C^{1,2}([0, T] \times R^d)$ function with bounded derivatives in x solution to

$$\begin{aligned} &\partial_t u(t, x) + Au(t, x) = f(x) \\ &u(T, x) = g(x) \end{aligned}$$

Then, if $Z = g(X_T) - \int_0^T f(X_s)ds$ and

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$Y = \int_0^T \partial_x u(s, X_s)\sigma(s, X_s)dW_s$$

we have

$$E[Z] = Z - Y$$

and Y is a perfect control variate of Z. Hence, we only need to look for H_t as a function of t and X_t. In a practical situation, we assume that we want to compute $u(0, x) = E[Z]$ with $Z = g(X_T) - \int_0^T f(X_s)ds$ and that we know a rough approximation $\overline{u}$ for u. We can therefore choose

$$Y = \int_0^T \partial_x \overline{u}(s, X_s)\sigma(s, X_s)dW_s \tag{26.2.6}$$

as a control variate. For every choice of $\overline{u}$ we obtain an unbiased estimator for $E[Z]$ by setting $\tilde{Z} = Z - Y$. For a reasonable choice of $\overline{u}$ we can expect an improvement of the variance of the estimator. If we set

$$\overline{Z} = g(X_T) - \int_0^T \partial_x \overline{u}(s, X_s)\sigma(s, X_s)dW_s$$

then $\overline{Z}$ is an unbiased estimator of $E[g(X_T)]$ and

$$E[|\overline{Z} - E[g(X_T)]|^2] = E\Big[\int_0^T |\partial_x u(t, X_s) - \partial \overline{u}(t, X_s)|^2 \sigma^2(s, X_s)ds\Big]$$

The variance is small if $\partial_x \overline{u}()$ is a good approximation of $\partial_x u()$. Hence, one approximates

$$\int_0^T \partial_x u(s, X_s)\sigma(s, X_s)dW_s$$

by the sum

$$\sum_{k=1}^n \partial_x \overline{u}(kh, X_{kh})\sigma(kh, X_{kh})(W_{(k+1)h} - W_{kh})$$

where $h = \frac{T}{n}$.

26.2.2.3 Using the hedge as control variates

The idea is to use an approximate gradient of the objective function as a control variate. For example, we let $(S_t)_{t\geqslant 0}$ be the state of the system with dynamics being

$$\frac{dS_t}{S_t} = rdt + \sigma(t, S_t)dW_t$$

and we let $C(t, S_t)$ be the objective value on that state. We assume that an explicit solution of that objective is known and given by $\overline{C}(t, S_t)$. Considering the control variate in Equation (26.2.6), it becomes

$$Y = \int_0^T \partial_x \overline{C}(s, S_s)\sigma(s, S_s)S_s dW_s = \int_0^T \partial_x \overline{C}(s, S_s)\big(dS_s - E[dS_s]\big)$$

Then we can use the random variable

$$Y = \sum_{k=1}^n \frac{\partial \overline{C}}{\partial x}(t_k, S_{t_k})\big((S_{t_{k+1}} - S_{t_k}) - E[S_{t_{k+1}} - S_{t_k}|\mathcal{F}_{t_k}]\big)$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

as control variate, which simplifies to

$$Y = \sum_{k=1}^{n} \frac{\partial \overline{C}}{\partial x}(t_k, S_{t_k})\big(S_{t_{k+1}} - E[S_{t_{k+1}}|\mathcal{F}_{t_k}]\big)$$

If $\overline{C}$ is close to C and if n is large enough, we can obtain a very large reduction of the variance. Given the solution

$$S_{t_{k+1}} = S_{t_k} e^{\int_{t_k}^{t_{k+1}} r_s ds} e^{-\frac{1}{2}\int_{t_k}^{t_{k+1}} \sigma^2(s,S_s)ds + \int_{t_k}^{t_{k+1}} \sigma(s,S_s)dW_s}$$

we get $E[S_{t_{k+1}}|S_{t_k}] = S_{t_k} e^{\int_{t_k}^{t_{k+1}} r_s ds}$.

26.2.2.4 Importance sampling

Here, we change the sampling law. Suppose we want to calculate $\mu = E_{x\sim f(x)}[g(X)]$ where X is a random variable with density $f(x)$ on $\mathbb{R}$. Then the expectation rewrite as

$$\begin{aligned} E_{x\sim f(x)}[g(X)] &= \int g(x)f(x)dx \\ &= \int g(x)\frac{f(x)}{\tilde{f}(x)}\tilde{f}(x)dx \\ &= E_{x\sim \tilde{f}(x)}[g(X)\frac{f(X)}{\tilde{f}(X)}] \end{aligned}$$

where X has density $\tilde{f}(x) > 0$ under $\mathcal{P}$. By making a multiplicative adjustment to g we compensate for sampling from $\tilde{f}$ instead of f. The adjustment factor $\frac{f(x)}{\tilde{f}(x)}$ is called the likelihood ratio. The distribution $\tilde{f}$ is the importance distribution, and f is the nominal distribution. The importance sampling estimate of μ is

$$\widehat{\mu}_{\tilde{f}} = \frac{1}{n}\sum_{i=1}^{n} g(X_i)\frac{f(X_i)}{\tilde{f}(X_i)}\ , X_i \sim \tilde{f}$$

Theorem 26.2.2 *Let $\widehat{\mu}_{\tilde{f}}$ be given above where $\mu = E_{x\sim f(x)}[g(X)]$ and $\tilde{f}(x) > 0$ whenever $g(x)f(x) \neq 0$. Then $E_{x\sim \tilde{f}(x)}[\widehat{\mu}_{\tilde{f}}] = \mu$ and $var_{\tilde{f}}(\widehat{\mu}_{\tilde{f}}) = \frac{\sigma^2_{\tilde{f}}}{n}$ where*

$$\begin{aligned} \sigma^2_{\tilde{f}} &= \int \frac{\big(g(x)f(x)\big)^2}{\tilde{f}(x)}dx - \mu^2 \\ &= \int \frac{\big(g(x)f(x) - \mu\tilde{f}(x)\big)^2}{\tilde{f}(x)}dx \end{aligned}$$

The variance is small when the numerator is close to zero, that is, when $\tilde{f}(x) \propto g(x)f(x)$. Thus, setting $\tilde{f}(x) = \frac{g(x)f(x)}{\mu}$, then $\sigma_{\tilde{f}} = 0$.
Denoting $Z = g(X)\frac{f(X)}{\tilde{f}(X)}$, then if

$$\tilde{f}(x) = \frac{g(x)f(x)}{E_{x\sim f(x)}[g(X)]}$$

we get $var(Z) = 0$.
To form a confidence interval for μ we need to estimate $\sigma^2_{\tilde{f}}$. From the second formula of the above theorem we have

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\sigma_{\tilde{f}}^2 = E_{x\sim\tilde{f}}[\frac{\big(g(X)f(X) - \mu\tilde{f}(X)\big)^2}{\tilde{f}^2(x)}]$$

Since x_i are sampled from $\tilde{f}$ the natural variance estimate is

$$\begin{aligned}\widehat{\sigma}_{\tilde{f}}^2 &= \frac{1}{n}\sum_{i=1}^{n}\big(\frac{g(x_i)f(x_i)}{\tilde{f}(x_i)} - \widehat{\mu}_{\tilde{f}}\big)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\big(w_i g(x_i) - \widehat{\mu}_{\tilde{f}}\big)^2\end{aligned}$$

where $w_i = \frac{f(x_i)}{\tilde{f}(x_i)}$. As an example, a 99% confidence interval for μ is $\widehat{\mu}_{\tilde{f}} \pm 2.58\frac{\sigma_{\tilde{f}}}{\sqrt{n}}$.

26.2.2.5 Antithetic variables

This technique is often efficient but its gains are less dramatic than other variance reduction techniques. If X is a random variable taking values in $\mathbb{R}^d$ and T is a transformation of $\mathbb{R}^d$ such that the law of $T(X)$ is the same as the law of X, we can construct an antithetic method using the equality

$$E[g(X)] = \frac{1}{2}E[g(X) + g(T(X))]$$

26.2.3 The speed of convergence

26.2.3.1 The Richardson extrapolation

The two-point Richardson extrapolation formula in the case of linear convergence is

$$\begin{aligned}P &= aP_k + bP_n \\ a &= \frac{k}{k-n} \text{ and } b = 1 - a\end{aligned}$$

where P_n is the price with n steps and p_k is the price with $k > n$ steps. In the special case where $k = 2n$ the extrapolation simplifies to

$$P = 2P_{2n} - P_n$$

In the case where we have square-root convergence, the two-point Richardson extrapolation formula becomes

$$\begin{aligned}P &= aP_k + bP_n \\ a &= \frac{\sqrt{k}}{\sqrt{k}-\sqrt{n}} \text{ and } b = 1 - a\end{aligned}$$

and in the special case where $k = 2n$ the extrapolation simplifies to

$$P = 3.414P_{2n} - 2.414P_n$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

26.2.3.2 The Romberg methods for Euler and Milstein scheme

A far better way to implement the convergence speed of a given scheme is the use of Romberg extrapolation method. It provides an approach to take optimally advantage of the weak rate of convergence, including in their higher oder form. To simplify, if the coordinates of b and σ are functions of class C^4 with bounded derivatives up to order 4, then there exist constants C_1 and C_2 independent of $h = \frac{T}{m}$ such that the Euler and Milstein schemes satisfy

$$|E[f(X_T)] - E[f(\hat{X}_{mh})]| \leqslant C_1 e^{C_2 T} h + \wr(h^2)$$

Talay et al. proved that the weak discretisation error of the Euler and Milstein schemes can be expanded in terms of $\frac{1}{m}$. We can summarise their results as follow. For all time steps $h = \frac{T}{m}$, the error at time T corresponding to the Euler scheme can be expanded as

$$E[f(X_T)] - E[f(\hat{X}_{mh})] = C(f)h + \wr(h^2)$$

where $C(f)$ is a constant which can be computed as a function of f. This expansion allow us to apply the Romberg-Richardson extrapolation technique. Put another way, we have

$$|E[f(X_T)] - (2E[f(\hat{X}_{mh})] - E[f(\hat{X}_{m2h})])| \leqslant K_T h^2$$

where in the first case $mh = T$ and in the second case $m2h = T$. The numerical cost of the Romberg-Richardson procedure is much smaller than the cost corresponding to schemes of order $\frac{1}{m^2}$. If f is smooth enough, it can be shown that for any integer $k > 0$ there exists constants $C_1, .., C_{k+1}$ depending on f such that

$$E[f(X_T)] - E[f(\hat{X}_{mh})] = \frac{C_1(f)}{m} + \frac{C_2(f)}{m^2} + ... + \frac{C_k(f)}{m^k} + \frac{R_{mh}}{m^{k+1}}$$

with $|R_{mh}|$ bounded. Going further in the analysis, if b and σ are functions of class C^4 with bounded derivatives up to order 4, and $f : \mathbb{R}^d \to \mathbb{R}$ is 4 times differentiable with polynomial growth, then

$$E[f(X_T^x)] - E[f(\hat{X}_T^x)] = \mathcal{O}(\frac{1}{m}) \text{ as } m \to \infty$$

Assuming more regularity on the coefficients or some uniform ellipticity on the diffusion coefficient σ one can obtain an expansion of the error at any order. Assume b and σ are infinitely differentiable with bounded partial derivatives, and $f : \mathbb{R}^d \to \mathbb{R}$ is infinitely differentiable with partial derivatives having polynomial growth. Then, given $h = \frac{T}{n}$ for every $R \geqslant 1$

$$\epsilon_{R+1} = E[f(\hat{X}_T)] - E[f(X_T)] = \sum_{k=1}^{R} \frac{c_k}{n^k} + \mathcal{O}(n^{-(R+1)}) \text{ as } n \to \infty$$

where the real coefficients c_k depend on f, T, b and σ. We let $f \in V$ where V is a vector space of continuous functions with linear growth. Given $X^1 = X$, we consider a regular MC simulation based on M copies $(\hat{X}_T)^m$ for $m = 1, .., M$ of the Euler scheme with step $h = \frac{T}{n}$ induces the global squared quadratic error

$$|E[f(X_T)] - \frac{1}{M}\sum_{m=1}^{M} f((\hat{X}_T)^m)|_2^2 = \frac{c_1^2}{n^2} + \frac{Var(f(\hat{X}_T))}{M} + \mathcal{O}(n^{-3})$$

We consider a second Brownian Euler scheme X^2 written with respect to a second Brownian motion W^2 defined on the same probability space $(\Omega, \mathcal{A}, \mathbb{P})$. We assume it has a twice smaller step $\frac{T}{2n}$ and is denoted $(\hat{X})^2$, then assuming (ϵ_2^V) to be more precise

$$E[f(X_T)] = E[2f((\hat{X}_T)^2) - f((\hat{X}_T)^1)] - \frac{1}{2}\frac{c_2}{n^2}\mathcal{O}(n^{-3})$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

we get the new global squared quadratic error

$$|E[f(X_T)] - \frac{1}{M}\sum_{m=1}^{M} 2f(((\hat{X}_T)^2)^m) - f(((\hat{X}_T)^1)^m)|_2^2 = \frac{c_2^2}{4n^4} + \frac{Var(2f((\hat{X}_T)^2) - f((\hat{X}_T)^1))}{M} + \mathcal{O}(n^{-5})$$

We need to find a way to control the term $Var(2f((\hat{X}_T)^2) - f((\hat{X}_T)^1))$. It is shown that if $W^2 = W^1 = W$, then

$$Var(2f(\hat{X}_T^2) - f(\hat{X}_T^1)) \to Var(2f(X_T) - f(X_T)) = Var(f(X_T)) \text{ as } n \to \infty$$

and that this choice is optimal among all possible choice of correlated Brownian motion W^1 and W^2. This result can be extended to Borel functions f when the diffusion is uniformly elliptic. Hence, one first simulates an Euler scheme with step $\frac{T}{2n}$ using a white Gaussian noise $(U_k^2)_{k\geqslant 1}$, then one simulates a Euler scheme with step $\frac{T}{n}$ using the white Gaussian noise

$$U_k^1 = \frac{U_{2k}^2 + U_{2k-1}^2}{\sqrt{2}}\,, k \geqslant 1$$

If one adopts the lazy approach based on independent Gaussian noises U^1 and U^2, then the asymptotic variance is

$$Var(2f(X_T^2) - f(X_T^1)) = 4Var(f(X_T^2)) + Var(f(X_T^1)) = 5Var(f(X_T^1))$$

A multi-step Romberg extrapolation with consistent Brownian increments was developed. In the case $R = 3$, we set

$$\alpha_1 = \frac{1}{2}\,, \alpha_2 = -4\,, \alpha_3 = \frac{9}{2}$$

such that

$$E\big[\alpha_1 f(\hat{X}_T^1) + \alpha_2 f(\hat{X}_T^2) + \alpha_3 f(\hat{X}_T^3)\big] = \frac{c_3^3}{n^3} + \mathcal{O}(n^{-4})$$

where $\hat{X}^r$ denotes the Euler scheme with step $\frac{T}{rn}$ for $r = 1, 2, 3$ with respect to the same Brownian motion W. Even though this choice induces a control of the variance of the estimator, it is no-longer optimal yet it is natural. The most efficient way to simulate the three white noises $(U_k^r)_{1\leqslant k\leqslant r}$ on one time step $\frac{T}{n}$ is to let U_i for $i = 1, 2, 3, 4$ be four i.i.d. copies of $N(0, I_q)$ and set

$$\begin{aligned}
&U_1^3 = U_1\,, U_2^3 = \frac{U_2 + U_3}{\sqrt{2}}\,, U_3^3 = U_4 \\
&U_1^2 = \frac{\sqrt{2}U_1 + U_2}{\sqrt{3}}\,, U_2^2 = \frac{U_3 + \sqrt{2}U_4}{\sqrt{3}} \\
&U_1^1 = \frac{U_1^2 + U_2^2}{\sqrt{2}}
\end{aligned}$$

26.2.4 Weak error for path-dependent functionals

We consider some path-dependent European options related to some payoffs $F((X_t)_{t\in[0,T]})$ where F is a functional defined on the set $\mathbb{D}([0,T],\mathbb{R}^d)$ of right continuous left-limited functions $x : [0,T] \to \mathbb{R}$. All the asymptotic control of the variance obtained in the previous section for the estimator $\sum_{r=1}^{R} \alpha_r f(\hat{X}_T^r)$ of $E[f(X_T)]$ when f is continuous can be extended to functionals $F : \mathbb{D}([0,T],\mathbb{R}^d) \to \mathbb{R}$ which are $\mathbb{P}_X$-a.s.continuous with respect to the sup-norm defined by $|x|_{sup} = \sup_{t\in[0,T]} |x(t)|$. This is because the continuous Euler scheme $\hat{X}$ with step $\frac{T}{n}$ defined by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\hat{X}_t = x_0 + \int_0^t b(\hat{X}_{\overline{s}})ds + \int_0^t \sigma(\hat{X}_{\overline{s}})dW_s \ , \ \overline{s} = \frac{ns}{T}$$

converges for the sup-norm toward X in every $L^P(\mathbb{P})$. Furthermore, the asymptotic control of the variance holds true with any R-tuple $\alpha = (\alpha_r)_{1 \leqslant r \leqslant R}$ of weights coefficients satisfying $\sum_{1 \leqslant r \leqslant R} \alpha_r = 1$ so that these coefficients can be adapted to the structure of the weak error expansion.

On the other hand, several papers provided some weak rates of convergence for some families of functionals F. For instance, in the case of barrier options

$$F(x) = \Phi(x(T))I_{\tau_D(x) \leqslant T}$$

where Φ is at least Lipschitz and $\tau_D = \inf\{s \in [0,T], x(s) \in D^c\}$ is the hitting time of D^c by x. For instance, if the domain D has smooth enough boundary, $b, \sigma \in C^3(\mathbb{R}^d)$, σ uniformly elliptic on D, then for every Borel bounded function f vanishing in a neighbourhood of ∂D then

$$E[f(\hat{X}_T)I_{\tau(\hat{X})>T}] - E[f(X_T)I_{\tau(X)>T}] = \mathcal{O}(\frac{1}{\sqrt{n}}) \text{ as } n \to \infty$$

If further, b and σ are C^5 then

$$E[f(\hat{X}_T)I_{\tau(\hat{X})>T}] - E[f(X_T)I_{\tau(X)>T}] = \mathcal{O}(\frac{1}{n}) \text{ as } n \to \infty$$

but these assumptions are not satisfied by usual barrier options. A similar improvement to $\mathcal{O}(\frac{1}{n})$ rate can be expected when replacing $\hat{X}$ by the continuous Euler scheme $\overline{X}$. For both classes of functionals, the practical implementation of the continuous Euler scheme is known as the Brownian Bridge method.

26.2.5 Romberg extrapolation for path-dependent functionals

There are two ways to implement the (multistep) Romberg extrapolation with consistent Brownian increments in order to improve the performance of the original (stepwise constant or continuous) Euler schemes. Both rely on natural conjectures about the existence of a higher order expansion of the time discretisation error suggested by the above rates of convergence.

26.2.5.1 The stepwise constant Euler scheme

For the stepwise constant Euler scheme, it means the existence of a vector space V (stable by product) of admissible functionals satisfying for all $F \in V$

$$(\epsilon_R^{\frac{1}{2},V}) \to E[F(X)] = E[F(\hat{X})] + \sum_{k=1}^{R-1} \frac{c_k}{n^{\frac{k}{2}}} + \mathcal{O}(n^{-\frac{R}{2}})$$

For small values of R, one gets

$$R = 2 \ \alpha_1^{\frac{1}{2}} = -(1+\sqrt{2}) \ , \ \alpha_2^{\frac{1}{2}} = \sqrt{2}(1+\sqrt{2})$$
$$R = 3 \ \alpha_1^{\frac{1}{2}} = \frac{\sqrt{3}-\sqrt{2}}{2\sqrt{2}-\sqrt{3}-1} \ , \ \alpha_2^{\frac{1}{2}} = -2\frac{\sqrt{3}-1}{2\sqrt{2}-\sqrt{3}-1} \ , \ \alpha_3^{\frac{1}{2}} = 3\frac{\sqrt{2}-1}{2\sqrt{2}-\sqrt{3}-1}$$

However, these coefficients have greater absolute values than in the standard case. Thus if $R = 4$, then $\sum_{1 \leqslant r \leqslant 4}(\alpha_r^{\frac{1}{2}})^2 \approx 10900!$ which induces an increase of the variance term for too small values of the time discretisation parameters n even when increments are consistently generated.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

26.2.5.2 The continuous Euler scheme

The conjecture is simply to assume that the expansion (ϵ_R) now holds for a vector space V of functionals F with polynomial growth with respect to the sup-norm. But the increase of the complexity induced by the Brownian Bridge method is difficult to quantise. It amounts to computing $\log U_k$ and the inverse distribution $F_{x,y}^{-1}$ and $G_{x,y}^{-1}$. Moreover, simulating the extrema of some continuous Euler schemes using the Brownian Bridge in a consistent way is not straightforward. But, one can reasonably expect that using independent Brownian Bridges relying on stepwise constant Euler schemes with consistent Brownian increments will have a small impact on the global variance (although slightly increasing it). So simulate the stepwise constant Euler scheme $\hat{X}^1$ and $\hat{X}^2$ and use consistent Brownian increments. Between two points t_k^n and t_{k+1}^n apply the BB to find the probability of crossing the barrier. Repeat the approach between the points t_k^{2n} and t_{k+1}^{2n}.

When the volatility is high, and $M = 10^6$ then the quasi-consistent 3-step Romberg extrapolation with Brownian Bridge clearly outperform the continuous Euler scheme (Brownian Bridge) of equivalent complexity while the 3-step Romberg extrapolation based on the stepwise constant Euler schemes with consistent Brownian increments is not competitive at all. It suffers from both a too high variance for the considered sizes of the MC simulation and from its too slow rate of convergence in time.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 27

Multifractal analysis

27.1 Introducing fractal analysis

27.1.1 A brief history

The Hurst Exponent (H) proposed by Hurst [1951] and tested by Hurst et al. [1965] relates to the autocorrelations of the time series, and the rate at which these decrease as the lag between pairs of values increases. It is used as a measure of long-term memory of time series, classifying the series as a random process, a persistent process, or an anti-persistent process. Hurst measured how a reservoir level (Nile River Dam project) fluctuated around its average level over time, and found that the range of the fluctuation would change, depending on the length of time used for measurement. If the series were random, the range would increase with the square root of time $T^{\frac{1}{2}}$. To standardise the measure over time, he created a dimensionless ratio by dividing the range by the standard deviation of the observation, obtaining the rescaled range analysis (R/S analysis). He found that most natural phenomena follow a biased random walk, that is, a trend with noise which could be measured by how the rescaled range scales with time, or how high the exponent H is above $\frac{1}{2}$.

In 1975 Mandelbrot (see Mandelbrot [1975] [2004]) introduced the term fractal as a geometric shape that is self-similar and has fractional dimensions, leading to the Mandelbrot set in 1977 and 1979. An object for which its topological dimension is lower or equal to its Hausdorff-Besicovitch (H-B) dimension is a fractal. By doing so, he was able to show how visual complexity can be created from simple rules. A more general definition states that a fractal is a shape made of parts similar to the whole in some way (see Feder [1988]). Two examples of fractal shapes are the Sierpinski Triange (see Figure (27.1)) and Koch Curve (see Figure (27.2)). The above Figures are bi-dimensional and deterministic fractal. Focusing on financial time series, Figure (27.3) illustrates a one-dimensional non-deterministic fractal representing thirty returns from the CAC40 Index at different time scales. Without any labels on the $X - Y$ scales, it is impossible to know which one is daily, weekly, or monthly returns. The fractal dimension D measures the smoothness of a surface, or, in our case, the smoothness of a time series. As discussed by Peters [1994], the importance of the fractal dimension of a time series is that it recognises that a process can be somewhere between deterministic (a line with fractal dimension of 1) and random (a fractal dimension of 1.50).

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

(a) Initial triangle (b) Second step (c) Third step (d) Few iterations

Figure 27.1: Sierpinski Triangle

(a) Initial line (b) Second step (c) Third step

(d) Few iterations

Figure 27.2: Koch Curve

Fractal analysis is done by conducting rescaled adjusted range analysis (R/S analysis) of time series (see Mandelbrot [**?**]), where the Hurst Exponent and the fractal dimension of the series are estimated. A major application of the R/S analysis is the study of price fluctuations in financial markets. There, the value of the Hurst Exponent, H, in a time series may be interpreted as an indicator of the irregularity of the price of a commodity, currency or similar quantity. Interval estimation and hypothesis testing for H are central to comparative quantitative analysis.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 27.3: Self-similarity in CAC40 Index returns

27.1.2 Presenting the results

Working on the Nile River Dam Project, Hurst was concerned with the storage capacity of a reservoir, based on an estimate of the water inflow and of the need for water outflow. Studying a 847-year record from 622 A.D. to 1469 A.D., he found that the record was not random, as larger than average overflows were more likely to be followed by more large overflows. Further, the process would abruptly change to a lower than average overflow, and be followed by more lower overflows. This is the characteristics of cycles with nonperiodic length. Working on Brownian motion, Einstein [1908] found that the distance R that a random particle covers increases with the square root of time T used to measure it, that is,

$$R = T^{\frac{1}{2}} \tag{27.1.1}$$

which is called the T to the one-half rule commonly used in statistics. It is used in financial economics to annualise volatility or standard deviation. However, in a sample data with N elements, we can only use this equation if the time series under consideration is independent of increasing values of N. That is, Equation (27.1.1) only applies to Brownian motion time series. Hurst generalised the concept to non-Brownian motion by taking into account systems that are not independent. Hurst et al. [1965] remedied this drawback by dividing the range by the standard deviation S of the original observations. They proposed a more general relation by calculating the Rescaled Range as

$$\frac{R}{S} = k \times T^H$$

where k is a constant depending on the time series (see Peters [1994] [1991-96]). The $\frac{R}{S}$ value has a mean of zero, is expressed in terms of local standard deviation S, and scales as we increase the time increment T by a power law value equals to the Hurst Exponent (H). By rescaling the data we can compare diverse phenomena and time periods. The value of the Hurst Exponent varies between 0 and 1, with $H = \frac{1}{2}$ implying a random walk or an independent process (see Figure 27.4). Figure (27.5) illustrates the role of the Hurst exponent. For $0 \leqslant H < \frac{1}{2}$ we have anti-persistence (or ergodicity) where the process covers less distance than a random walk. It is often referred as mean reverting, meaning that if the system has been up in the previous period, it is more likely to be down in the next period. The converse being true. Anti-persistent time series exhibit higher noise and more volatility than a random series because it consists of frequent reversals. For $\frac{1}{2} < H \leqslant 1$ we have persistence (or trend-reinforcing) where the process covers more distance than a random walk. A persistent series has long memory effects such that the trend at a particular point in time affects the remainder of the time series. If the series has been up (down) in the last period, then the chances are that it will continue to be positive (negative) in the next period.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Figure 27.4: Range of Hurst exponents between 0.1 and 0.9

(a) $H = 0.1$ (b) $H = 0.5$ (c) $H = 0.9$

Figure 27.5: Cumulative observations for different values of H

The fractal dimension D of a time series measures how jagged the time series is. In order to calculate the fractal dimension of a time series we count the number of circles of a given, fixed diameter needed to cover the entire time

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

series. We then increase the diameter and count again the number of circles required to cover the time series. Repeating that process, the number of circles has an exponential relationship with the radius of the circle satisfying

$$N \times d^D = 1$$

where N is the number of circles, and d is the diameter. Transforming the above equation [1], the fractal dimension is given by

$$D = \frac{\log(N)}{\log(\frac{1}{d})} \tag{27.1.2}$$

Thus, the fractal dimension of a time series is a function of scaling in time. Mandelbrot showed that the Hurst Exponent, H, is related to the fractal dimension D for a self-similar surface in n-dimensional space by the relation

$$D = n + 1 - H \tag{27.1.3}$$

where $0 \leqslant H \leqslant 1$. While the fractal dimension of a line is 1.0 and the fractal dimension of a geometric plane is 2.0, the fractal dimension of a random walk is half way between the line and the plane at 1.5. A value of $\frac{1}{2} < H \leqslant 1$ results in a fractal dimension closer to a line, so that a persistent time series would result in a smoother, less jagged line than a random walk. Similarly, for $0 < H < \frac{1}{2}$, a time series is more jagged than a random series, or has more reversals.

Definition 27.1.1 *The Hurst Exponent is the measure of the smoothness of fractal time series based on the asymptotic behaviour of the rescaled range of the process.*

The Hurst Exponent can be estimated by

$$H = \frac{\log(\frac{R}{S})}{\log(T)}$$

where T is the duration of the sample data, and $\frac{R}{S}$ is the corresponding value of the rescaled range.

Definition 27.1.2 *The Rescaled Range is the measure characterising the divergence of time series defined as the range of the mean-centred values for a given duration divided by the standard deviation of that duration.*

Persistent series are fractional Brownian motion, or biased random walk with the strength of the bias depending on how far H is above $\frac{1}{2}$. For $H \neq \frac{1}{2}$ each observation carry a long-term memory, theoretically lasting forever. A system exhibiting Hurst statistics is the result of a long stream of interconnected events where time is important as the future depends on the past. As discussed in Feder [1988], the variance of fractionally integrated processes is proportional to t^{2H}, and the impact of the present on the future can be expressed as a correlation

$$C = 2^{(2H-1)} - 1 \tag{27.1.4}$$

where C is the correlation measure. For $H = \frac{1}{2}$ we get $C = 0$, and events are random and uncorrelated. The R/S analysis can classify an independent series, no matter what is the shape of the underlying distribution. Note, the correlation C is not related to the Auto Correlation Function (ACF) of Gaussian random variables which assumes Gaussian, or near-Gaussian, properties in the underlying distribution.

[1]
$$N = (\frac{1}{d})^D$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

27.2 Defining random fractals

Mandelbrot et al. [1968] introduced the fractional Brownian motion (fBm) as a generalisation of the ordinary Brownian motion by considering correlations between the increments of the process. The fBm and its corresponding generalised derivative process, the fractional Gaussian noise (fGn), are the benchmark of the Fractal Market Hypothesis (FMH). This is the only family of processes being Gaussian, self-similar, and endowed with stationary increments.

27.2.1 The fractional Brownian motion

We consider the one dimensional random process $X(t)$ with values $X(t_1), X(t_2), ...$ such that the increment $X(t_2) - X(t_1)$ has a Gaussian distribution, and the mean square increments have a variance proportional to the time differences,

$$E[|X(t_2) - X(t_1)|^2] \propto |t_2 - t_1|$$

We say that the increments of X are statistically self-similar in the sense that

$$X(t_0 + t) - X(t_0) \text{ and } \frac{1}{\sqrt{r}}(X(t_0 + rt) - X(t_0))$$

have the same finite dimensional joint distribution functions for any t_0 and time scale $r > 0$. The integral of uncorrelated white Gaussian noise W satisfies the above equations, and is given by

$$X(t) = \int_{-\infty}^{t} W(s)ds$$

where the random variables $W(t)$ are uncorrelated and have the same normal distribution. The fractional Brownian motion (fBm) is a generalisation satisfying

$$Var(X(t_2) - X(t_1)) \propto |t_2 - t_1|^{2H}$$

for $0 < H < 1$. In the special case $H = \frac{1}{2}$ we recover the Brownian motion. Again, the increments of X are statistically self-similar with parameter H in the sense that

$$X(t_0 + t) - X(t_0) \text{ and } \frac{1}{r^H}(X(t_0 + rt) - X(t_0))$$

have the same finite dimensional joint distribution functions for any t_0 and $r > 0$. Setting $t_0 = 0$, the accelerated fBm $X(rt)$ is properly rescaled by dividing amplitudes by r^H.

More formally, the fractional Brownian motions, $\{B_H(t), t > 0\}$, is a Gaussian process with zero mean and stationary increments whose variance and covariance are given by

$$\begin{aligned} E[B_H^2(t)] &= \sigma^2 t^{2H} \\ E[B_H(s)B_H(t)] &= \frac{1}{2}\sigma^2\left(s^{2H} + t^{2H} - |t-s|^{2H}\right), t, s > 0 \end{aligned}$$

where $0 < H < 1$, $\sigma > 0$. Note, fBm are continuous but non-differentiable (in the classical sense). Note, Mandelbrot et al. [1968] defined the above as self-affinity of increments in distribution and proposed the following definition of self-similarity in distribution

Definition 27.2.1 *A process $(X_t)_{-\infty<t<\infty}$ is called self-similar if*

$$X(at) \to a^H X(t)$$

for a positive factor a and non-negative self-similarity parameter H.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

A major consequence of this definition is that the moments of X, provided they exist, behave as power laws of time

$$E[|X(t)|^q] = E[|X(1)|^q]|t|^{qH} \tag{27.2.5}$$

In the framework of stochastic processes, such laws could only hold in distribution. In this case, Mandelbrot et al. speak of self-affine processes. The self-affinity in distribution is defined as

Definition 27.2.2 *A process $(X_t)_{-\infty<t<\infty}$, $X(0) = 0$, is called self-affine if*

$$\{X(ct_1), ..., X(ct_k)\} \to \{c^H X(t_1), .., c^H X(t_k)\}$$

for non-negative factors c and k, periods $t_1, .., t_k$, and a positive self-similarity parameter H.

so that self-affinity in distribution is a special case of self-similarity in distribution. Definition (27.2.1) determines the dependency structure of the increments of a process obeying such scaling behaviour as well as their higher moments showing hyperbolic decline of their autocorrelations with an exponent depending linearly on H.

27.2.2 The multidimensional fBm

A fractional Brownian function $f(r) : \mathbb{R}^d \to \mathbb{R}$ is charactersised by

- a variance
$$\sigma_H^2 =< [f(r+\lambda) - f(r)]^2 > \propto \|\lambda\|^\alpha \text{ , } \alpha = 2H$$
with $r = (x_1, .., x_d)$, $\lambda = (\lambda_1, .., \lambda_d)$, and $\|\lambda\| = \sqrt{\lambda_1^2 + ... + \lambda_d^2}$.
- a power spectrum
$$S_H \propto \|w\|^{-\beta} \text{ , } \beta = d + 2H$$
with $w = (w_1, .., w_d)$ the angular frequency, and $\|w\| = \sqrt{w_1^2 + ... + w_d^2}$.
- a number of objects N_H of characteristic size ϵ needed to cover the fractal
$$N_H \propto \epsilon^{-D} \text{ , } D = d + 1 - H$$
where D is the fractal dimension of $f(r)$,

In the special case where $d = 1$, the power spectrum for the fractional Brownian motion becomes

$$S_H \propto \frac{1}{|w|^\beta}$$

with $\beta = 2H + 1$ and $1 < \beta < 3$. Similarly, the power spectrum for the fractional Gaussian noise becomes

$$S_H \propto \frac{1}{|w|^\beta}$$

with $\beta = 2H - 1$ and $-1 < \beta < 1$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

27.2.3 The fractional Gaussian noise

The implications of self-similarity for time series are not based on distributions but on dynamic properties of the time series which are defined by the autocorrelation function $\gamma(k)$ (see Section ()). We denote by $\{W_H(t), t > 0\}$ the process derived from the increment of fBm, namely

$$W_H(t) = B_H(t+1) - B_H(t)$$

Beran [1994] and Embrechts et al. [2002] showed that self-similar processes have autocorrelation function defined exactly and asymptotically in the following two propositions.

***Proposition* 47** *Let $(X_t)_{t=0}^T$ be self-similar process with $0 < H < 1$ and finite variance $\sigma^2 < \infty$. Then the correlations are given by autocorrelation function*

$$\gamma(k) = E[X_t X_{t+k}] = \frac{(k+1)^{2H} - 2k^{2H} + (k-1)^{2H}}{2}$$

for $k \geqslant 0$, where H is the Hurst exponent.

***Proposition* 48** *Let $(X_t)_{t=0}^T$ be self-similar process with $0 < H < 1$ and finite variance $\sigma^2 < \infty$. Then the correlations are asymptotically given by autocorrelation function*

$$\gamma(k) \sim H(2H-1)k^{2H-2}$$

for $k \to \infty$, where H is the Hurst exponent.

Note, the second Proposition (48) suggests that if $H = \frac{1}{2}$, we have either an independent process, if the first Proposition (47) holds as well, or a short-term dependent process, if that first proposition is not valid. Thus, an independent process has zero correlations at all non-zero lags, while a short-term dependent process has significantly non-zero correlations at low lags but zero correlations at high lags. For $H > \frac{1}{2}$, the process has significantly positive correlations at all lags. It has hyperbolically decaying correlations which are non-summable and $\sum_{k=0}^{\infty} \gamma(k) = \infty$. For $H < \frac{1}{2}$, the process has significantly negative correlations at all lags decaying hyperbolically. However, the correlations are summable, and we get $0 < \sum_{k=0}^{\infty} \gamma(k) < \infty$ (see Embrechts et al. [2002]). The spectral density (Fourier transform of γ) is

$$f(\lambda) = C_H(2\sin(\frac{\lambda}{2}))^2 \sum_{k=-\infty}^{\infty} \frac{1}{|\lambda + 2\pi k|^{2H+1}} \sim C_H|\lambda|^{1-2H} \text{ as } \lambda \to 0$$

where C_H is a constant.

27.2.4 The fractal process and its distribution

Following Mandelbrot et al. [1997] we now define fractal process with the separation between multi-fractal and monofractal processes of Lux [2003].

Definition 27.2.3 *A process $(X_t)_{-\infty<t<\infty}$ is called fractal if*

$$X(ct) \to M(c)X(t) = c^{H(c)}X(t) \tag{27.2.6}$$

for a positive factor c and an arbitrary non-negative function $H(c)$. If $H(c) = H$ is a constant function, the process is said to be monofractal. Otherwise, the process is said to be multi-fractal.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The scaling factor $M(c)$ is a random function with possibly different shape for different scales. One moves from one scale to another via multiplication with a random factor $M(c)$. The last equality emphasise the fact that this variability of scaling laws can be translated into variability of the exponent H. Note, a self-affine process is a special case of a self-similar process which is in turn a special case of a fractal process, all connected by the parameter H showing persistent, anti-persistent, short-term dependent and independent processes. Levy distributions are stable distributions. According to Levy, a distribution function, $F(x)$, is stable if, for all $b_1, b_2 > 0$, there also exists $b > 0$ such that

$$F(\frac{x}{b_1}) \times F(\frac{x}{b_2}) = F(\frac{x}{b})$$

This is a general characteristic of the class of stable distribution. The characteristic functions of F can be expressed in a similar manner

$$f(b_1 t) \times f(b_2 t) = f(bt)$$

so that $f(b_1 t)$, $f(b_2 t)$, and $f(bt)$ all have the same shaped distribution, despite their being products of one another which accounts for their stability. Following Mandelbrot [?], we are now going to define the family of stable distributions, also called fractal because of their self-similar properties.

Definition 27.2.4 *Stable distributions are determined by characteristic function, natural logarithm of which is defined as*

$$\phi(t) = \ln E[e^{ixt}] = i\delta t - |ct|^\alpha \big(1 - i\beta \frac{t}{|t|} \tan \frac{\pi\alpha}{2}\big)$$

for $\alpha \neq 1$ *and*

$$\phi(t) = i\delta t - |ct|^\alpha \big(1 + i\beta \frac{2}{\pi} \ln |t|\big)$$

for $\alpha = 1$*, where* $0 < \alpha \leqslant 2$ *is a characteristic exponent determining peakedness at* δ*,* $\beta \leqslant |1|$ *is a skewness parameter,* $-\infty < \delta < \infty$ *is a location parameter and* $0 \leqslant c < \infty$ *is a scale parameter.*

Note, $\frac{t}{|t|}$ is the the sign of t. We usually represent the parameters of a stable distribution by $S(x; \alpha, \beta, c, \delta)$. For $\alpha = 2$, $\beta = 0$ we recover the Gaussian distribution with variance equal to $c = 2\sigma^2$. Further, if $\delta = 0$, we get the standardise normal distribution with zero mean and unit variance. Setting $\alpha = 1$, $\beta = 0$, we recover the Cauchy distribution which has infinite variance and mean. The parameter α is crucial for an existence of variance. For $1 < \alpha < 2$, the distribution has infinite or undefined second moment and thus population variance. Moreover, for $0 < \alpha \leqslant 1$, the distribution has infinite mean as well. Even though there are a number of different ways of measuring the characteristic exponent α (see Fama [1965]), the R/S analysis and spectral analysis offer the most reliable method. Peters [1991-96], Panas [2001] and others showed that the self-similar process is connected to stable distribution of the process by the following relation between the Hurst exponent and the characteristic exponent α

$$\alpha = \frac{1}{H} \tag{27.2.7}$$

While H measures the fractal dimension of the time trace by the fractal dimension $2 - H$ (see Equation (27.1.3) with $n = 1$), it is also related to the statistical self-similarity of the process through Equation (27.2.7). However, $\frac{1}{H}$ measures the fractal dimension of the probability space.

27.3 Defining multifractality

Multifractality or anomalous scaling allows for a richer variation of the behaviour of a process across different scales. Equation (27.2.6) shows that this variability of scaling laws can be translated into the variability of the Hurst index,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

which is no-longer constant. Following Abry et al. [2002] and Wendt [2008] who both gave a synthetic overview of the concepts of scale invariance, scaling analysis, and multifractal analysis, we are going to introduce multifractality.

27.3.1 Fractal measures and their singularities

In nonlinear physics, one want to characterise complicated fractal objects and describe the events occurring on them. For example, this characterisation applies to dynamical systems theory, diffusion-limited aggregation, percolation, to name a few. In general, one describe such events by dividing the object into pieces labelled by index i running from 1 to N. The size of the ith piece is l_i and the event occurring upon it is given by the number M_i. For instance, in the droplet theory of Ising model, the magnetisation has a value of the order of

$$M_i \sim l^y$$

where y is a critical index. Since these droplets should fill the entire space, the density of such droplets is

$$\rho(l) \sim \frac{1}{l^d} \tag{27.3.8}$$

where d is the Euclidean dimension of space. Halsey et al. [1986] considered the case where each M_i is a probability that some event will occur upon the ith piece. They assumed the d-dimensional time series $\{X_i\}_{i=1}^N$ with trajectories lying on a strange attractor of dimension D, $D < d$ and tried to find out how many times N_i would the time series visit the ith box. They defined $p_i = \lim_{N\to\infty} \frac{N_i}{N}$ and generated the measure on the attractor $d\mu(X)$, since $p_i = \int_{\text{ith box}} d\mu(X)$. That is, p_i is the probability (integrated measure) in the ith box. If a scaling exponent α [2] is defined by

$$p_i^q \sim l_i^{\alpha q}$$

then α can take on a range of values, corresponding to different regions of the measure. They suggested that the number of times α would take on value between $\alpha^{'}$ and $\alpha^{'} + d\alpha^{'}$ should be of the form

$$d\alpha^{'} \rho(\alpha^{'}) l^{-f(\alpha^{'})}$$

where $f(\alpha^{'})$ is a continuous function. It reflects the differing dimensions of the sets upon which the singularities of strength $\alpha^{'}$ may lie. It is roughly equivalent to Equation (27.3.8) with the dimension d replaced by the fractal dimension $f(\alpha)$ which varies with α. Thus, fractal measures are modelled by interwoven sets of singularities of strength α, each characterised by its own dimension $f(\alpha)$.

The function $f(\alpha)$ must then be related to observable properties of the measure. The most basic property of a strange attractor is its dimension, and the three main ones are the fractional (or similarity) dimension D of the support of the measure (see also box counting dimension), the information dimension σ, and the correlation dimension ν. Hentschel et al. [1983] showed these attractors are characterised by an infinite number of generalised dimension D_q, $q > 0$. Rather than considering droplet theory of Ising model, they let $\{X_i\}_{i=1}^N$ be the points on the attractor covering space with a mesh of d-dimensional cubes of size b^d, and $M(b)$ be the number of cubes containing points of the series. They let $p_\alpha = \frac{N_\alpha}{N}$ be the probability to fall in one of the boxes of type α and defined the rescaling hierarchy based on a box of size l^d on the nth level covered with bins of size b^d where the probability of the ith bin is denoted by $p_i(l, b)$. They showed self-similarity between the box of size l^d on the nth level of the hierarchy and the box of size $((\frac{l}{s})_\alpha)^d$ on the $(n+1)$th level which is used to calculate the dimensions associated with the natural measure. So far, the exponent ν was related to correlations between pairs of point on the fractal, but Hentschel et al. [1983] considered higher order correlation functions or correlation integrals $C_n(l)$ and showed that

$$C_n(l) \approx C_n(l, b) = \sum_{i\in l} p_i^n(l, b)$$

[2] Lipschitz-Holder exponent

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

scales like

$$C_n(l,b) = C_n(l)b^{\nu_n}$$

Obtaining D_n for an integer n, they generalised it to an uncountable infinity of quantities D_q with any $q > 0$ defined by

$$D_q = \lim_{l \to 0}\Big(\frac{1}{q-1}\frac{\ln \chi(q)}{\ln l}\Big) \tag{27.3.9}$$

where

$$\chi(q) = \sum_{i \in l} p_i^q \tag{27.3.10}$$

is a partition function. Hence, the generalised dimensions D_q correspond to the scaling exponents for the qth moments of the measure. Note, D_0 is the fractional (similarity) dimension, D_1 is the information dimension, and D_2 is the correlation dimension. That is, multifractals have multiple dimension in the D_q versus q spectra, but monofractals stay rather flat in that area. Note, the term

$$S_q = \frac{1}{q-1}\ln \chi(q)$$

is called the Renyi entropy representing a one-parametric generalisation of Shannon's entropy (for which it reduces for $q \to 1$).

Later, Halsey et al. [1986] related the function $f(\alpha)$ to the set of dimensions D_q, and after replacing p_i with its definition above, they obtained

$$\chi(q) = \int d\alpha^{'} \rho(\alpha^{'}) l^{-f(\alpha^{'})} l^{q\alpha^{'}} = \int d\alpha^{'} \rho(\alpha^{'}) l^{q\alpha^{'} - f(\alpha^{'})}$$

Since l is very small, this integral is dominated by the value of $\alpha^{'}$ making $q\alpha^{'} - f(\alpha^{'})$ smallest, provided that $\rho(\alpha^{'})$ is nonzero. To find the minimum, they replaced $\alpha^{'}$ by $\alpha(q)$, which is defined by the external condition

$$\frac{d}{d\alpha^{'}}\big[q\alpha^{'} - f(\alpha^{'})\big]\Big|_{\alpha^{'}=\alpha(q)} = 0$$

Differentiating one more time, we have

$$\frac{d^2}{d(\alpha^{'})^2}\big[q\alpha^{'} - f(\alpha^{'})\big]\Big|_{\alpha^{'}=\alpha(q)} > 0$$

such that $f^{'}(\alpha(q)) = q$ and $f^{''}(\alpha(q)) < q$.

***Remark* 27.3.1** *As a result of the optimisation process, $\alpha(q)$ is the value for which the expression $q\alpha - f(\alpha)$ is extremal.*

Then, from the definition of the generalised fractal dimensions in Equation (27.3.9), they obtained

$$D_q = \frac{1}{q-1}\big[q\alpha(q) - f(\alpha(q))\big] \tag{27.3.11}$$

so that knowing $f(\alpha)$ and the spectrum of α values, one can define D_q. Alternatively, given D_q, one can get $\alpha(q)$ [3] since

[3] $\frac{d}{dq}\big[q\alpha(q) - f(\alpha(q))\big] = \alpha(q) + q\frac{d}{dq}\alpha(q) - \frac{d}{d\alpha}f(\alpha)\frac{d}{dq}\alpha(q) = \alpha(q)$ and $f^{'}(\alpha(q)) = q$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\alpha(q) = \frac{d}{dq}(q-1)D_q$$

Consequently, the generalised dimensions D_q provide an alternative description of the singular measure. Generalising the definition of the dimension D_q, they considered a strange set S embedded in a finite portion of d-dimensional Euclidean space, and partitioned the set into some number of disjoint pieces $S_1, .., S_N$ in which each piece has a measure p_i and lies within a ball of radius l_l, where each l_i is restricted by $l_i < l$. Then given the partition function

$$\Gamma(q, \tau, \{S_i, l\}) = \sum_{i=1}^{N} \frac{p_i^q}{l_i^\tau}$$

they argued that for large N, it is of the order unity only when $\tau = (q-1)D_q$. Defining

$$\Gamma(q, \tau) = \lim_{l \to 0} \Gamma(q, \tau, l)$$

and arguing that the function $\tau(q)$ is unique, they defined D_q as

$$(q-1)D_q = \tau(q) \tag{27.3.12}$$

which allows to recover $\alpha(q)$ and $f(\alpha(q))$. That is, replacing in Equation (27.3.11), we get

$$\tau(q) = \left[q\alpha(q) - f(\alpha(q))\right] \tag{27.3.13}$$

and

$$\alpha(q) = \frac{d}{dq}\tau(q) \tag{27.3.14}$$

Note, $\tau(q)$ is the Legendre transform of $f(\alpha)$. Further, this definition of D_q precisely makes D_0 the Hausdorff dimension. They considered some simple examples to illustrate the quantities $\tau(q)$ and gain intuition about $\alpha(q)$ and $f(\alpha)$.

This multifractal formalism (MF), which is very useful in the characterisation of singular measures, accounts for the statistical scaling properties of these measures through the determination of their singularity spectrum $f(\alpha)$ which is intimately related to the generalised fractal dimension D_q. It reflects a deep connection with the thermodynamic formalism (TF) of equilibrium statistical mechanics where $\tau(q)$ and q are conjugate theremodynamic variables to $f(\alpha)$ and α. In fact, the concept of multifractality originated from a general class of multiplicative cascade models introduced by Mandelbrot [1974] in the context of fully developed turbulence (FDT). In the past, the D_q has usually been used on real, or, computer experiments, and the $f(\alpha)$ curves were determined from the Legendre transform of the $\tau(q)$ curves, which involved first smoothing the former and then Legendre transforming. A few years later, Chhabra et al. [1989] proposed a direct evaluation of $f(\alpha)$ without resorting to the intermediate Legendre transform. Denoting the probabilities in the boxes of size l as $p_j(l)$, they constructed a one-parameter family of normalised measure $\mu(q)$ given by

$$\mu_i(q, l) = \frac{p_i^q(l)}{\sum_j p_j^q(l)} \tag{27.3.15}$$

and they obtained the singularity spectrum as

$$\begin{aligned} f(q) &= \lim_{l \to 0} \frac{\sum_i \mu_i(q,l) \ln \mu_i(q,l)}{\ln l} \\ \alpha(q) &= \lim_{l \to 0} \frac{\sum_i \mu_i(q,l) \ln p_i(l)}{\ln l} \end{aligned} \tag{27.3.16}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

which provides a relationship between the Hausdorff dimension f and an average singularity strength α as an implicit function of the parameter q. We are now going to discuss how one can estimate the $f(\alpha)$ spectrum based on the local dissipation.

27.3.2 Scaling analysis

As explained in Section (27.3.1), processes with spectra obeying a power law within a given (and sufficiently wide) range of frequencies (scales) are often refered to as $\frac{1}{f}$ processes

$$\Gamma_X(\nu) = C_0|\nu|^{-\gamma}\,,\, \nu_m \leqslant |\nu| \leqslant \nu_M$$

where $\Gamma_X(\nu)$ is any standard spectrum estimation procedure, $0 < \gamma < 1$ and C_0 is a positive constant. Two special cases where the scale range is semi-infinite, either at small frequencies, $\nu_m \to 0$ (equivalently large scales) or at large frequencies, $\nu_M \to \infty$ (small scales), define long-range dependent processes and monofractal processes. Scale invariance is generally defined as the power law behaviours of (the time average of the qth power of) multiresolution quantities, denoted $T_X(a,t)$ with respect to the analysis scale a, for a given (large) range of scales $a \in (a_m, a_M)$, $\frac{a_M}{a_m} >> 1$

$$\frac{1}{n_a}\sum_{k=1}^{n_a}|T_X(a,k)|^q \simeq c_q a^{\tau(q)} \tag{27.3.17}$$

where $T_X(a,k)$ describes the content of X around a time position t, and a scale a, such as a wavelet transform (see Muzy et al. [1991], Abry et al. [1995]). As explained in Section (27.3.1), $\tau(q)$ is a scaling exponent related to the generalised dimension via Equation (27.3.12). Hence, these quantities can be seen as some kind of spectral estimates.

Definition 27.3.1 *Scaling analysis*
The aim of scaling analysis is to validate the existence of power law behaviours as in Equation (27.3.17), and to measure the scaling exponents $\tau(q)$ that characterise them.

As suggested by Equation (27.3.17), the analysis and estimation procedures consist in tracking straight lines and estimating slopes in log-log plots. The estimated exponents can then be used for the physical understanding of the data or the system producing them. Note, the central theoretical notion of scale invariance is that of self-similarity, where the statistical information obtained from an object is independent of the scale of observation. However, self-similarity is a very demanding property, as it implies exact invariance to dilatation for all scale factors $a > 0$ of all finite dimensional distributions, therefore involving all statistical orders of the process. In practice, investigation of scale invariance is often restricted to the statistical order 2, or only certain range of scale factors. To be more flexible, one definition of scale invariance is obtained by relaxing the self-similarity property. First, scale invariance has to hold only for a restricted range of scaling factors $a \in (a_m, a_M)$, $\frac{a_M}{a_m} >> 1$, rather than all $a > 0$. Second, the (single) self-similarity parameter H is replaced with the function $\tau(q)$, called the scaling function or the scaling exponents of $X(t)$.

Definition 27.3.2 *Scale invariant stationary process*
Suppose $X(t)$ is a process with stationary increments. Then it is scale invariant if

$$E[|X(at)|^q] = |a|^{\tau(q)}E[|X(t)|^q]$$

for a range of scale $a \in (a_m, a_M)$, $\frac{a_M}{a_m} >> 1$ and some range of statistical orders q.

which implies for the increment process

$$E[|\delta_\theta X(t)|^q] = |\theta|^{\tau(q)}E[|\delta_1 X(0)|^q]\,,\, 0 < \theta_m < \theta < \theta_M < \infty$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $\delta_\theta X(t) = X(t+\theta) - X(t)$ is a longitudinal velocity increment over a distance θ. The single parameter H is replaced with a function $\tau(q)$, which is in general non-linear $(\tau(q) \neq qH)$, and hence represents a whole collection of parameters for the characterisation of the process.

While the work of Frisch et al. [1985] and Halsey et al. [1986], described in Section (27.3.1), showed that there was an infinite hierarchy of exponents allowing for a much more complete representation of the fractal measures, the related ideas of multifractality have only been applied to self-similar sets. Barabasi et al. [1991] extended the concept of multifractality to self-affine fractals, providing a more complete description of fractal surfaces. They investigated the multiscaling properties of the self-affine function $f(X)$ by calculating the qth order height-height correlation function

$$C_q(\Delta t) = \frac{1}{N}\sum_{t=1}^{N} |X(t,\Delta t)|^q$$

where $X(t,\Delta t) = X(t) - X(t-\Delta t)$, $N >> 1$ is the number of points over which the average is taken, and only the terms with $|X(t,\Delta t)| > 0$ are considered. They showed that this correlation function exhibited a nontrivial multiscaling behaviour if

$$C_q(\Delta t) \sim (\Delta t)^{qH(q)} \tag{27.3.18}$$

with $H(q)$ changing continuously with q at least for some region of the q values. As a result, multiscaling in empirical data is typically identified by differences in the scaling behaviour of different (absolute) moments

$$E[|X(t,\Delta t)|^q] = c(q)(\Delta t)^{qH(q)} = c(q)(\Delta t)^{\tau(q)} \tag{27.3.19}$$

where $c(q)$ and $\tau(q)$ are deterministic functions of the order of the moment q. From Equation 27.3.19 we can see that $\tau(q) = qH(q)$, which motivates the definition of the generalised Hurst exponent as

$$H(q) = \frac{\tau(q)}{q} \tag{27.3.20}$$

A similar expression holds for mono-scaling processes such as fBm

$$E[|X(t,\Delta t)|^q] = c^H(\Delta t)^{qH} \tag{27.3.21}$$

***Remark* 27.3.2** *There exists different ways of defining the notion of scale invariance and multifractality in the literature.*

In terms of the behaviour of moments, monofractal, self-affine processes, are characterised by linear scaling, while multifractality (anomalous scaling) is characterised by non-linear (typically concave) shape. Hence, to diagnose multifractality, we can consider the inspection of the empirical scaling behaviour of an ensemble of moments. The traditional approach in the physics literature consists in extracting the self-similarity exponent, $\tau(q)$, from a chain of linear log-log fits of the behaviour of the various moments q for a certain selection of time aggregation steps Δt. We can therefore use regressions to the temporal scaling of moments of powers q

$$\ln E[|X(t,\Delta t)|^q] = a_0 + a_1 \ln(\Delta t) \tag{27.3.22}$$

and constructs the empirical $\tau(q)$ curve (for a selection of discrete q) from the ensemble of estimated regression coefficients for all q. An alternative and more widespread approach consists in looking directly at the varying scaling coefficients $H(q)$. This is called multifractal analysis which we introduce in Section (27.3.3). While the unique coefficient H quantifies a global scaling property of the underlying process, the multiplicity of such coefficients in multifractal processes, called Holder exponents, can be viewed as local scaling rates governing various patches of a time series, leading to a characteristically heterogeneous (or intermittent) appearance of such series. Focusing on the

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

concept of Holder exponents, multifractality analysis amounts to identifying the range of such exponents rather than a degenerate single H as in the case of monofractal processes.

27.3.3 Multifractal analysis

So far, we focused on global properties such as moments and autocovariance, we are now going to adopt a more local viewpoint and examine the regularity of realised paths around a given instant. Contrary to scaling analysis which concentrate on the scaling exponent $\tau(q)$, the multifractal analysis (MFA) studies how the (pointwise) local regularity of X fluctuates in time (or space). Local Holder regularity describes the regularity of sample paths of stochastic processes by means of a local comparison against a power law function and is therefore closely related to scaling in the limit of small scales. The exponent of this power law, $h(t)$, is called the (local) Holder exponent and depends on both time and the sample path of X.

Definition 27.3.3 *Pointwise regularity*
A function $f(t)$, $f : \mathbb{R}^d \to \mathbb{R}$ is $C^\alpha(t_0)$ with $\alpha > 0$, denoted as $f \in C^\alpha(t_0)$, if there exist $C > 0$, $\epsilon > 0$ and a polynomial $P_{t_0}(\theta)$ of order strictly smaller than α such that

$$\text{if } |\theta| \leqslant \epsilon \,,\, |f(t_0+\theta) - P_{t_0}(\theta)| \leqslant C|\theta|^\alpha$$

If such polynomial $P_{t_0}(\theta)$ exists, it is unique, and its constant part is always given by $P_{t_0}(0) = f(t_0)$.

Definition 27.3.4 *Holder exponent*
The Holder exponent $h_f(t_0)$ of f at t_0 is

$$h_f(t_0) = \sup\{\alpha : f \in C^\alpha(t_0)\}$$

Note, if $P_{t_0}(0) = f(t_0)$ reduces to a constant, the Holder exponent characterises the power law behaviour of the increments at t_0

$$|f(t_0+\theta) - f(t_0)| \leqslant C(t_0)|\theta|^{h_f(t_0)}$$

where $C(t_0)$ is called the prefactor at time t_0. Put another way, the exponent $h_f(t_0)$ quantifies the scaling properties of the process at time t_0. The Holder exponent generalises the heuristic definition of singularity, so that it is also called the singularity exponent. That is, if f has Holder exponent $h(t_0) = h_f(t_0) < 1$ at t_0, then f has at t_0 either a cusp-type singularity

$$|f(t_0+\theta) - f(t_0)| \sim C|\theta|^{h(t_0)} \tag{27.3.23}$$

or an oscillating singularity

$$|f(t_0+\theta) - f(t_0)| \sim C|\theta|^{h(t_0)} \sin\left(\frac{1}{|\theta|^\beta}\right) \tag{27.3.24}$$

with oscillation exponent $\beta > 0$. Conversely, if f has either a cusp or an oscillating singularity at t_0 and $1 > h(t_0)$, then $h(t_0) = h_f(t_0)$ is the Holder exponent of f at t_0.

Multifractal situations happen when the Holder exponent is no-longer unique, but vary from point to point. More precisely, when the regularity $h(t)$ is itself a highly irregular function of t, possibly even a random process rather than a constant or a fixed deterministic function, the process X is said to be multifractal. The aim of multifractal analysis is to provide a description of the collection of Holder exponents h of the function f. Since the Holder exponent may jump from one point to another, it was understood that describing them for each time step was not of much importance. Hence, researchers focused on a global description of the regularity of the function of f in form of

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

multifractal spectrum (also called the singularity spectrum) reflecting the size of the set of points for which the Holder exponent takes a certain value h. The measure of size most commonly used is the Hausdorff dimension which gives rise to the Hausdorff spectrum (called the multifractal spectrum). It describes the collection of Holder exponents $h(t)$ by mapping to each value of h the Hausdorff dimension $D(h)$ of the collection of points t_i at which $h_f(t_i) = h$. The main idea being that the relative frequency of the local exponents can be represented by a renormalised density called the multifractal spectrum.

Definition 27.3.5 *Iso-Holder sets*
The Iso-Holder sets $I_f(h)$ is the collection of points t_i for which the Holder exponent takes a certain value h.

$$I_f(h) = \{t_i | h_f(t_i) = h\}$$

For defining the Hausdorff dimension, we first need to define the Hausdorff measure.

Definition 27.3.6 *Hausdorff measure*
Let $S \subset \mathbb{R}^d$, $\epsilon > 0$, and let $\gamma_\epsilon(S)$ be ϵ-coverings of S, that is, bounded sets $\{c_n\}_{n\in\mathbb{N}}$ of radius $|c_n| \leqslant \epsilon$ (maximal distance between two elements of c_n) that cover $S : S \subset \gamma_\epsilon(S)$. Let $C_\epsilon(S)$ be the collection of all ϵ-coverings $\gamma_\epsilon(S)$ of S. The δ-dimensional Hausdorff measure of S is

$$m_\delta(S) = \lim_{\epsilon\to 0} \inf_{C_\epsilon(S)} \sum_{\gamma_\epsilon(S)} |c_n|^\delta$$

It can be shown that either $m_\delta(S) = 0$ if $\delta < \delta_c$, or $m_\delta(S) = \infty$ if $\delta > \delta_c$. The Hausdorff dimension of S is defined as the critical value δ_c.

Definition 27.3.7 *Hausdorff dimension*
The Hausdorff dimension $dim_H(S)$ of $S \subset \mathbb{R}^d$ is given by

$$dim_H(S) = \inf_\delta\{m_\delta(S) = \infty\} = \sup_\delta\{m_\delta(S) = 0\}$$

The multifractal spectrum assigns now to each Holder exponent, as a measure of its geometric importance, the Hausdorff dimension of the set of points that share the same exponent h.

Definition 27.3.8 *Multifractal spectrum*
The multifractal spectrum of a function f is defined as the Hausdorff dimension of the iso-Holder sets $I_f(h)$.

$$D_f(h) = dim_H(I_f(h))$$

We can now review the definition of monofractal and multifractal processes.

Definition 27.3.9 *Monofractal function or process*
A function or process $X(t)$ in $\mathbb{R}^d$ for which $h_X(t)$ is a constant, $\forall t : h_X(t) = H$, is called monofractal.

Definition 27.3.10 *Multifractal function or process*
A function or process $X(t)$ in $\mathbb{R}^d$ is called multifractal if it contains more than one Holder exponent h that is living on a support with non-zero Hausdorff dimension.

All homogeneous processes X with Holder exponents $h_X(x)$ not constant have their exponents which are discontinuous everywhere and hence are highly variable and change widely from point to point. As pointed out by Abry et al. [2002], one of the major consequences of multifractality in processes lies in the fact that quantities called partition functions present power law behaviours in the limit of small scales

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$S_\theta(q) = \sum_{k=1}^{\frac{1}{\theta}} |\delta_\theta X(k\theta)|^q \simeq c(q)|\theta|^{\tau(q)-1}\,, |\theta| \to 0$$

which correspond to Equation (27.3.18) with $N = \frac{1}{\theta}$. For processes with stationary increments, the time averages $\theta S_\theta(q)$ can be seen as estimators for the statistical averages $E[|\delta_\theta X(t)|^q]$ for $t = k\theta$. As a result, the above equation is highly reminiscent of the fundamental Equation (27.2.5) implied by self-similarity. However, the exponents $\tau(q)$ need not to follow the linear behaviour qH of self-similarity.

27.3.4 The wavelet transform and the thermodynamical formalism

Considering the multifractal spectrum of a process to be the multifractal spectrum of each of his realisations, we can speak of multifractal spectrum for both functions and processes. Hence, multifractal analysis determine the multifractal spectrum which describe a local property, the point-wise regularity of a function, globally through the geometrical importance of different Holder exponents, disregarding any information on their precise geometric repartition. Following Parisi et al. [1985], under some assumptions on the homogeneity and isotropy of the statistics of local singularities, it is possible to derive a relation between self-similarity exponents $\tau(q)$ and the singularity spectrum $D(h)$. They showed that the self-similarity exponents $\tau(q)$ could be computed from the the Legendre transformation [4] of the singularity spectrum $D(h)$

$$\tau(q) = \inf_h \{qh + d - D(h)\} \tag{27.3.25}$$

where d is the dimension of the embedding space. This is to be related to the work on fractal measures by Halsey et al. [1986] and described in Section (27.3.1).

***Remark* 27.3.3** *It correspond to Equation (27.3.11) with Holder exponent satisfying $\alpha' = h$, that is, before it has been minimised. Once it has been minimised we can replace h with $h(q)$. We also see that the multifractal spectrum $D(h)$ corresponds to the singularity spectrum $f(\alpha(q))$.*

Using this equation, one can then relate statistics and geometry. Note, the singularity spectrum is often estimated by using indirect methods which are numerically more stable, such as the Legendre spectrum. We can invert the equation above since the Legendre spectrum correspond to the Legendre transform of the self-singularity exponents $\tau(q)$. That is, the spectrum of Holder exponents (or multifractal spectrum, or Legendre spectrum) can be obtained by the Legendre transformation of the scaling function $\tau(q)$ as

$$D_L(h) = \inf_q \{qh + d - \tau(q)\}$$

which can be computed numerically since there exists a lot of estimation procedure for the scaling function $\tau(q)$. For example, Calvet et al. [2002] reported the spectrum of multiplicative measures when the random variable V is binomial, Poisson, or Gamma. Note, this method failed to fully characterise the singularity spectrum $D(h)$, since only the strongest singularities are amenable to this analysis. In order to determine the whole singularity spectrum $D(h)$, Muzy et al. [1991] considered the wavelet transform (WT) and established the foundations for a thermodynamical formalism (TF) for singular signals. In that setting, the singularity spectrum $D(h)$ is directly determined from the scaling behaviour of partition functions defined from the wavelet transform modulus maxima (WTMM). Improving on the standard partition multifractal formalism, the wavelet transform modulus maxima (WTMM) is based on wavelet analysis, and involves tracing the maxima lines in the continuous wavelet transform over all scales (see Arneodo et al. [1995]). Later, under the uniform Holder regularity condition for X, Wendt [2008] showed that $D_L(h)$ with $\tau_L(q)$ estimated from the wavelet Leader structure functions

[4] The Legendre transformation is a mathematical operation transforming a function of a coordinate, $g(x)$, into a new function $h(y)$ whose argument is the derivative of $g(x)$ with respect to x, i.e., $y = \frac{dg(x)}{dx}$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$S^L(j,q) = \frac{1}{n_j} \sum_{k=1}^{n_j} L_X^q(j,k)$$

where $L_X(j,k)$ are wavelet Leader coefficients, provided a tight upper bound to the multifractal spectrum $D(h)$

$$D(h) \leqslant D_L(h)$$

***Remark* 27.3.4** *The structure function $S^L(j,q)$ is a special case of the time average of the qth power of multiresolution quantities $T_X(a,t)$ in Equation (27.3.17), where $T_X(a,t)$ has been replaced with the wavelet Leader coefficients. We will see that other representations of the quantities $T_X(a,t)$ exist, defining alternative models.*

The wavelet leader multifractal formalism (WLMF) asserts that this inequality turns into an equality

$$\forall h \, , D(h) = D_L(h) = \inf_q \{qh + d - \tau(q)\}$$

so that the Legendre spectrum of X can be interpreted in terms of the Holder singularities of X (see Wendt [2008]). One can then compute numerically the Legendre spectrum $D_L(h)$. Hence, the scaling exponents $\tau(q)$ and the multifractal spectrum $D(h)$ are closely related via the Legendre transform. That is, the power law behaviours together with the multifractal spectrum constitute the fundamental relations establishing the connection between scale invariance and multifractality.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 28

Portfolio Theory

28.1 Miscellaneous

28.1.1 The capital asset pricing model

In the problem of portfolio selection, the mean-variance approach introduced by Markowitz [1952] is a simple trade-off between return and uncertainty, where one is left with the choice of one free parameter, the amount of variance acceptable to the individual investor. For proofs and rigorous introduction to the mean-variance portfolio technique see Huang et al. [1988]. For a retrospective on Markowitz's portfolio selection see Rubinstein [2002]. Investment theory based on growth is an alternative to utility theory with simple goal. Following this approach, Kelly [1956] used the role of time in multiplicative processes to solve the problem of portfolio selection.

28.1.1.1 Mean-variance criterion

28.1.1.1.1 Normal returns We assume that an investor has an exponential utility (EU) function with a Coefficient of Absolute Risk Aversion (CARA) γ given by

$$u(W) = -e^{-\gamma W}\,, \gamma > 0$$

and we further assume that the returns on a portfolio are normally distributed with expectation μ and standard deviation σ. Hence, the certain equivalent (CE) of the portfolio can be approximated as

$$CE \approx \mu - \frac{1}{2}\gamma\sigma^2$$

The expected portfolio return is

$$\mu = w^\top E[r]$$

where w is the vector of portfolio weights and r is the vector of returns on the constituent assets, and the portfolio variance is

$$\sigma^2 = w^\top Q w$$

where Q is the covariance matrix of the asset returns. Since the best investment is the one giving the maximum certain equivalent, then for an investor with an exponential utility function investing in risky assets with normally distributed returns, the optimal allocation problem may be approximated by the simple optimisation

$$\max_w \left(\mu - \frac{1}{2}\gamma\sigma^2\right) = \max_w \left(w^\top E[r] - \frac{1}{2}\gamma w^\top Q w\right) \tag{28.1.1}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

which is the mean-variance criterion. Note, when the utility is defined on the returns on the investment, we must multiply the utility function by the amount invested to find the utility of each investment. Similarly, to find the certain equivalent of a risky investment we multiply by the amount invested. However, we need to express the CRA γ as a proportion of the amount invested.

28.1.1.1.2 Non-normal returns Assuming that investors borrow at zero interest rate, we consider the utility associated with an investment as being defined on the distribution of investment returns rather than on the distribution of the wealth arising from the investment. We further assume that the returns are non-normally distributed. Applying the expectation operator to a Taylor expansion of $u(R)$ about $u(\mu)$, the utility associated with the mean return, we get

$$E[u(R)] = u(\mu) + u^{'}(R)|_{R=\mu}E[R-\mu] + \frac{1}{2}u^{''}(R)|_{R=\mu}E[(R-\mu)^2] + \frac{1}{6}u^{'''}(R)|_{R=\mu}E[(R-\mu)^3] + ...$$

which is a simple approximation to the certain equivalent associated with any utility function since $E[u(R)] = E[u(X)] = u(CE(X))$.
If we assume that the investor has an exponential utility (EU) function, then, the above equation to the fourth order becomes

$$e^{-\gamma CE} \approx e^{-\gamma\mu}\big(1 + \frac{1}{2}\gamma^2 E[(R-\mu)^2] - \frac{1}{6}\gamma^3 E[(R-\mu)^3] + \frac{1}{24}\gamma^4 E[(R-\mu)^4]\big)$$

Given the first four moments are

$$\sigma^2 = E[(R-\mu)^2]\,, S = E[(R-\mu)^3]\,, K = E[(R-\mu)^4]$$

where S is the skew and K is the kurtosis, we get the approximation

$$e^{-\gamma CE} \approx e^{-\gamma\mu}\big(1 + \frac{1}{2}(\gamma\sigma)^2 - \frac{S}{6}(\gamma\sigma)^3 + \frac{K-3}{24}(\gamma\sigma)^4\big)$$

where $K-3$ is the excess kurtosis. Taking the logarithm and using the second order Taylor expansion, the approximated certain equivalent (CE) associated with the exponential utility function simplifies to

$$CE \approx \mu - \frac{1}{2}\gamma\sigma^2 + \frac{S}{6}\gamma^2\sigma^3 - \frac{K-3}{24}\gamma^3\sigma^4$$

The mean-variance criterion is a special case of the above equation with no skewness and no kurtosis.

***Remark* 28.1.1** *In general, a risk averse investor having an exponential utility has aversion to risk associated with increasing variance, negative skewness, and increasing kurtosis.*

28.1.1.2 Markowitz solution to the portfolio allocation problem

A rational investor should allocate his funds between the different risky assets in his universe, leading to the portfolio allocation problem. To solve this problem Markowitz [1952] introduced the concept of utility functions to express investor's risk preferences. Markowitz first considered the rule that the investor should maximise discounted expected, or anticipated returns (which is linked to the St. Petersurg paradox). However, he showed that the law of large numbers can not apply to a portfolio of securities since the returns from securities are too intercorrelated. That is, diversification can not eliminate all variance. Hence, rejecting the first hypothesis, he then considered the rule that the investor should consider expected return a desirable thing and variance of return an undesirable thing.

The mean-variance efficient portfolios are obtained as the solution to a quadratic optimisation program. Its theoretical justification requires either a quadratic utility function or some fairly restrictive assumptions on the class of return distribution, such as the assumption of normally distributed returns.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

For instance, we assume zero transaction costs and portfolios with prices V_t taking values in $\mathbb{R}$ and following the geometric Brownian motion with dynamics under the historical probability measure $\mathbb{P}$ given by

$$\frac{dV_t}{V_t} = \mu dt + \sigma_V dW_t \tag{28.1.2}$$

where μ is the drift, σ_V is the volatility and W_t is a standard Brownian motion. Markowitz first considered the problem of maximising the expected rate of return

$$g = \frac{1}{dt} < \frac{dV_t}{V_t} >= \mu$$

(also called ensemble average growth rate) and rejected such strategy because the portfolio with maximum expected rate of return is likely to be under-diversified, and as a result, to have an unacceptable high volatility. As a result, he postulated that while diversification would reduce risk, it would not eliminate it, so that an investor should maximise the expected portfolio return μ while minimising portfolio variance of return σ_V^2. It follows from the relation between the variance of the return of the portfolio σ_V^2 and the variance of return of its constituent securities σ_j^2 for $j = 1, 2, ..., N$ given by

$$\sigma_V^2 = \sum_j w_j^2 \sigma_j^2 + \sum_j \sum_{k \neq j} w_j w_k \rho_{jk} \sigma_j \sigma_k$$

where the w_j are the portfolio weights such that $\sum_j w_j = 1$, and ρ_{jk} is the correlation of the returns of securities j and k. Therefore, $\rho_{jk}\sigma_j\sigma_k$ is the covariance of their returns. So, the decision to hold any security would depend on what other securities the investor wants to hold. That is, securities can not be properly evaluated in isolation, but only as a group. Consequently, Markowitz suggested calling portfolio i efficient if

1. there exists no other portfolio j in the market with equal or smaller volatility, $\sigma_j \leqslant \sigma_i$, whose drift term μ_j exceeds that of portfolio i. That is, for all j such that $\sigma_j \leqslant \sigma_i$, we have $\mu_j \leqslant \mu_i$.
2. there exists no other portfolio j in the market with equal or greater drift term, $\mu_j \geqslant \mu_i$, whose volatility σ_j is smaller than that of portfolio i. That is, for all j such that $\mu_j \geqslant \mu_i$, we have $\sigma_j \geqslant \sigma_i$.

In the presence of a riskless asset (with $\sigma_i = 0$), all efficient portfolios lie along a straight line, the efficient frontier, intersecting in the space of volatility and drift terms, the riskless asset r_f and the so-called market portfolio M. Since any point along the efficient frontier represents an efficient portfolio, additional information is needed in order to select the optimal portfolio. For instance, one can specify the usefulness or desirability of a particular investment outcome to a particular investor, namely his risk preference, and represent it with a utility function $u = u(V_t)$. Following the work of von Neumann et al. [1944] and Savage [1954], Markowitz [1959] found a way to reconcile his mean-variance criterion with the maximisation of the expected utility of wealth after many reinvestment periods. He advised using the strategy of maximising the expected logarithmic utility of return each period for investors with a long-term horizon, and developed a quadratic approximation to this strategy allowing the investor to choose portfolios based on mean and variance.

28.1.1.3 The expected growth rate

As an alternative to the problem of portfolio selection, Kelly [1956] proposed to maximise the expected growth rate

$$g^b = \frac{1}{dt} < d \ln V_t >= \mu - \frac{1}{2}\sigma^2$$

obtained by using Ito's formula (see Peters [2011c]). This rate is called the expected growth rate, or the logarithmic geometric mean rate of return (also called the time average growth rate). In that setting, we observe that large returns and small volatilities are desirable. Note, Ito's formula changes the behaviour in time without changing the noise

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

term. That is, Ito's formula encode the multiplicative effect of time (for noise terms) in the ensemble average (see Oksendal [1998]). Hence, it can be seen as a mean of accounting for the effects of time. For self-financing portfolios, where eventual outcomes are the product over intermediate returns, maximising g^b yields meaningful results. This is because it is equivalent to using logarithmic utility function $u(V_t) = \ln V_t$. In that setting, the rate of change of the ensemble average utility happens to be the time average of the growth rate in a multiplicative process. Note, the problem that additional information is needed to select the right portfolio disappears when using the expected growth rate g^b. That is, there is no need to use utility function to express risk preferences as one can use solely the role of time in multiplicative processes.

28.1.2 Risk and return analysis

Asset managers employ risk metrics to provide their investors with an accurate report of the return of the fund as well as its risk. Risk measures allow investors to choose the best strategies per rebalancing frequency in a more robust way. Performance evaluation of any asset, strategy, or fund tends to be done on returns that are adjusted for the average risk taken. We call active return and active risk the return and risk measured relative to a benchmark. Since all investors in funds have some degree of risk aversion and require limits on the active risk of the funds, they consider the ratio of active return to active risk in a risk adjusted performance measure (RAPM) to rank different investment opportunities. In general, RAPMs are used to rank portfolios in order of preference, implying that preferences are already embodied in the measure. However, for an agent to make a decision, he needs a utility function. While some RAPMs have a direct link to a utility function, others are still used to rank investments but we can not deduce anything about preferences from their ranking so that no decision can be based on their ranks (see Alexander [2008]).

The three measures by which the risk/return framework describes the universe of assets are the mean (taken as the arithmetic mean), the standard deviation, and the correlation of an asset to other assets' returns. Concretely, historical time series of assets are used to calculate the statistics from it, then these statistics are interpreted as true estimators of the future behaviour of the assets. In addition, following the central limit theorem, returns of individual assets are jointly normally distributed. Thus, given the assumption of a Gaussian (normal) distribution, the first two moments suffice to completely describe the distribution of a multi-asset portfolio. As a result, adjustment for volatility is the most common risk adjustment leading to Sharpe type metrics. Implicit in the use of the Sharpe ratio (SR) is the assumption that the preferences of investors can be represented by the exponential utility function. This is because the tractability of an exponential utility function allows an investor to form optimal portfolios by maximising a mean-variance criterion (see Appendix (28.1.1.1)). However, some RAPMs are based on downside risk metrics which are only concerned with returns falling short of a benchmark or threshold returns, and are not linked to a utility function. Nonetheless these metrics are used by practitioners irrespectively of their theoretical foundation. For more details see Bloch [2014].

28.1.2.1 Performance measures

When considering the performance evaluation of mutual funds, one need to assess whether these funds are earning higher returns than the benchmark returns (portfolio or index returns) in terms of risk. Three measures developed in the framework of the Capital Asset Pricing Model (CAPM) proposed by Treynor [1965], Sharpe [1964] and Lintner [1965] directly relate to the beta of the portfolio through the security market line (SML). Jensen's [1968] alpha is defined as the portfolio excess return earned in addition to the required average return, while the Treynor ratio and the Information ratio are defined as the alpha divided by the portfolio beta and by the standard deviation of the portfolio residual returns. More recent performance measures developed along hedge funds, such as the Sortino ratio, the M2 and the Omega, focus on a measure of total risk, in the continuation of the Sharpe ratio applied to the capital market line (CML). In the context of the extension of the CAPM to linear multi-factor asset pricing models, the development of measures has not been so prolific (see Hubner [2007]).

The Sharpe ratio or Reward to Variability and Sterling ratio have been widely used to measure commodity trading advisor (CTA) performance. One can group investment statistics as Sharpe type combining risk and return in a ratio,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

or descriptive statistics (neither good nor bad) providing information about the pattern of returns. Examples of the latter are regression statistics (systematic risk), covariance and R^2. Additional risk measures exist to accommodate the risk concerns of different types of investors. Some of these measures have been categorised in Table (28.1).

Table 28.1: List of measure

Type	Combined Return and Risk Ratio
Normal	Sharpe, Information, Modified Information
Regression	Apraisal, Treynor
Partial Moments	Sortino, Omega, Upside Potential, Omega-Sharpe, Prospect
Drawdown	Calmar, Sterling, Burke, Sterling-Calmar, Pain, Martin
Value at Risk	Reward to VaR, Conditional Sharpe, Modified Sharpe

28.1.2.1.1 The Sharpe ratio The Sharpe ratio (SR) measures the excess return per unit of deviation in an investment asset or a trading strategy defined as

$$M_{SR} = \frac{E[R_a - R_b]}{\sigma} \tag{28.1.3}$$

where R_a is the asset return and R_b is the return of a benchmark asset such as the risk free rate or an index. Hence, $E[R_a - R_b]$ is the expected value of the excess of the asset return over the benchmark return, and σ is the standard deviation of this expected excess return. It characterise how well the return of an asset compensates the investor for the risk taken. If we graph the risk measure with a the measure of return in the vertical axis and the measure of risk in the horizontal axis, then the Sharpe ratio simply measures the gradient of the line from the risk-free rate to the combined return and risk of each asset (or portfolio). Thus, the steeper the gradient, the higher the Sharpe ratio, and the better the combined performance of risk and return.

The SR assumes that assets are normally distributed or equivalently that the investors' preferences can be represented by the quadratic (exponential) utility function. That is, the portfolio is completely characterised by its mean and volatility (see Appendix (28.1.1.1)). As soon as the portfolio is invested in technology stocks, distressed companies, hedge funds or high yield bonds, this ratio is no-longer valid. In that case, the risk comes not only from volatility but also from higher moments like skewness and kurtosis. Abnormalities like kurtosis, fatter tails and higher peaks or skewness on the distribution can be problematic for the computation of the ratio as standard deviation does not have the same effectiveness when these problems exist. As a result, we can get very misleading measure of risk-return.

28.1.2.2 Incorporating tail risk

While the problem of fat tails is everywhere in financial risk analysis, there is no solution, and one should only consider partial solutions. Hence, one practical strategy for dealing with the messy but essential issues related to measuring and managing risk starts with the iron rule of never relying on one risk metric. Even though all of the standard metrics have well-known flaws, that does not make them worthless, but it is a reminder that we must understand where any one risk measure stumbles, where it can provide insight, and what are the possible fixes, if any.

The main flaw of the Sharpe ratio (SR) is that it uses standard deviation as a proxy for risk. This is a problem because standard deviation, which is the second moment of the distribution, works best with normal distributions. Therefore, investors must have a minimal type of risk aversion to variance alone, as if their utility function was exponential. Even though normality has some validity over long periods of time, in the short run it is very unlikely (see short maturity smile on options). Note, when moving away from the normality assumption for the stock returns, only the denominator in the Sharpe ratio is modified. Hence, all the partial solutions are attempts at expressing one way

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

or another the proper noise of the stock returns. While extensions of the SR to normality assumption have been successful, extension to different types of utility function have been more problematic.

The problem is that extreme losses occur more frequently than one would expect when assuming that price changes are always and forever random (normally distributed). Put another way, statistics calculated using normal assumption might underestimate risk. One must therefore account for higher moments to get a better understanding of the shape of the distribution of returns in view of assessing the relative qualities of portfolios. Investors should prefer high average returns, lower variance or standard deviation, positive skewness, and lower kurtosis. The adjusted Sharpe ratio suggested by Pezier et al. [2006] explicitly rewards positive skewness and low kurtosis (below 3, the kurtosis of a normal distribution) in its calculation

$$M_{ASR} = M_{SR}\big[1 + \frac{S}{6}M_{SR} - \frac{K-3}{24}M_{SR}^2\big]$$

where S is the skew and K is the kurtosis. This adjustment will tend to lower the SR if there is negative skewness and positive excess kurtosis in the returns. Hence, it potentially removes one of the possible criticisms of the Sharpe ratio. Hodges [1997] introduced another extension of the SR accounting for non-normality and incorporating utility function. Assuming that investors are able to find the expected maximum utility $E[u^*]$ associated with any portfolio, the generalised Sharpe ratio (GSR) of the portfolio is

$$M_{GSR} = \big(-2\ln\left(-E[u^*]\right)\big)^{\frac{1}{2}}$$

One can avoid the difficulty of computing the maximum expected utility by assuming the investor has an exponential utility function. Using the fourth order Taylor approximation of the certain equivalent, and approximating the multiplicative factor Pezier et al. [2006] obtained the maximum expected utility function in that setting and showed that the GSR simplifies to the ASR. Thus, when the utility function is exponential and the returns are normally distributed, the GSR is identical to the SR. Otherwise a negative skewness and high positive kurtosis will reduce the GSR relative to the SR.

28.1.2.3 Considering the value at risk

28.1.2.3.1 Introducing the value at risk The Value at Risk (VaR) is a widely used risk measure of the risk of loss on a specific portfolio of financial assets. VaR is defined as a threshold value such that the probability that the mark-to-market loss on the portfolio over the given time horizon exceeds this value (assuming normal markets and no trading in the portfolio) is the given probability level. For example, if a portfolio of stocks has a one-day 5% VaR of $\$1$ million, there is a 0.05 probability that the portfolio will fall in value by more than $\$1$ million over a one day period if there is no trading. Informally, a loss of $\$1$ million or more on this portfolio is expected on 1 day out of 20 days (because of 5% probability). VaR represents a percentile of the predictive probability distribution for the size of a future financial loss. That is, if you have a record of portfolio value over time then the VaR is simply the negative quantile function of those values.

We let r_P be the return of the portfolio, f_{r_P} is the probability density function (pdf) of r_P, and F_R is the cumulative distribution function (cdf) of r_P. We denote the VaR of the portfolio for a probability level p as $VaR(F_{r_P}, p)$. When expressed as a percentage of the initial value of the portfolio and as a positive number, the VaR of the portfolio can be expressed as

$$VaR(F_{r_P}, p) = -F_{r_P}^{-1}(1-p)$$

That is, given a confidence level $\alpha \in (0,1)$, the VaR of the portfolio at the confidence level α is given by the smallest number z_p such that the probability that the loss L exceeds z_p is at most $(1-\alpha)$. Assuming normally distributed returns, the Value-at-Risk (daily or monthly) is

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$VaR(p) = W_0(\mu - z_p\sigma)$$

where W_0 is the initial portfolio wealth, μ is the expected asset return (daily or monthly), σ is the standard deviation (daily or monthly), and z_p is the number of standard deviation at $(1-\alpha)$ (distance between μ and the VaR in number of standard deviation). It ensures that

$$P(dW \leqslant -VaR(p)) = 1 - \alpha$$

Note, $VaR(p)$ represents the lower bound of the confidence interval given in Appendix (26.2.1). For example, setting $\alpha = 5\%$ then $z_p = 1.96$ with $p = 97.5$ which is a 95% probability.

If returns do not display a normal distribution pattern, the Cornish-Fisher expansion can be used to include skewness and kurtosis in computing value at risk (see Favre et al. [2002]). It adjusts the z-value of a standard VaR for skewness and kurtosis as follows

$$z_{cf} = z_p + \frac{1}{6}(z_p^2 - 1)S + \frac{1}{24}(z_p^3 - 3z_p)K - \frac{1}{36}(2z_p^3 - 5z_p)S^2$$

where z_p is the critical value according to the chosen α-confidence level in a standard normal distribution, S is the skewness, K is the excess kurtosis. Integrating them into the VaR measure by means of the Cornish-Fisher expansion z_{cf} , we end up with a modified formulation for the VaR, called MVaR

$$MVaR(p) = W_0(\mu - z_{cf}\sigma)$$

28.1.2.3.2 The reward to VaR We saw earlier that when the risk is only measured with the volatility it is often underestimated, because the assets returns are negatively skewed and have fat tails. One solution is to use the value-at-risk as a measure of risk, and consider Sharpe type measures using VaR. For instance, replacing the standard deviation in the denominator with the VaR ratio (Var expressed as a percentage of portfolio value rather than an amount) Dowd [2000] got the Reward to VaR

$$M_{RVaR} = \frac{r_P - r_F}{\text{VaR ratio}}$$

Note, the VaR measure does not provide any information about the shape of the tail or the expected size of loss beyond the confidence level, making it an unsatisfactory risk measure.

28.1.2.3.3 The conditional Sharpe ratio Tail risk is the possibility that investment losses will exceed expectations implied by a normal distribution. One attempt at trying to anticipate non-normality is the modified Sharpe ratio, which incorporates skewness and kurtosis into the calculation. Another possibility is the so-called conditional Sharpe ratio (CSR) or expected shortfall, which attempts to quantify the risk that an asset or portfolio will experience extreme losses. VaR tries to tell us what the possibility of loss is up to some confidence level, usually 95%. So, for instance, one might say that a certain portfolio is at risk of losing $X\%$ for 95% of the time. What about the remaining 5%? Conditional VaR, or CVaR, dares to tread into this black hole of fat taildom (by accounting for the shape of the tail). It measures the expectation of the losses greater than or equal to the VaR and is given by the ratio of the size of the losses beyond the VaR to the frequency of losses greater than or equal to the VaR. It can be expressed as

$$CVaR(F_{r_P}, p) = -E[r_P | r_P \leqslant -VaR] = -\frac{\int_{-\infty}^{-VaR} x f_{r_P}(x)dx}{F_{r_P}(-VaR)}$$

For the conditional Sharpe ratio, CVaR replaces standard deviation in the metric's denominator

$$M_{CVaR} = \frac{r_P - r_F}{CVaR(p)}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The basic message in conditional Sharpe ratio, like that of its modified counterpart, is that investors underestimate risk by roughly a third (or more?) when looking only at standard deviation and related metrics (see Agarwal et al. [2004]).

28.2 Dynamical models

We will follow the notation from Dana et al. [1994].

28.2.1 The discrete model

Following Harrison et al. [1979] [1981], we consider a finite horizon model with $(d+1)$ assets.

28.2.1.1 Description

We let $(\Omega, \mathcal{F}, P)$ be a probability space with the filtration $(\mathcal{F}_n; n = 0, ..., N)$. We consider an $\mathcal{F}_n$-adapted sequence of random variables $(S_n; n \leqslant N)$. We assume $(d+1)$ assets $S_n = (S_0(n), S_1(n), .., S_d(n))$ with values in $\mathbb{R}_+^{d+1}$ are traded at instant n. The asset S_0 is risk-free and given by $S_0(n) = (1+r)^n S_0(0)$ where r is the spot rate. We assume $S_0(0) = 1$ and S_n is $\mathcal{F}_n$-adapted. We let the portfolio strategy θ_n^i corresponds to the quantity invested in the ith asset at instant n. Then, $\pi_n^i = \theta_n^i S_n^i$ is the portion of wealth invested in the ith asset at instant n. We let $\theta = (\theta_n; n = 1, ..., N)$ be a family of random vectors such that

$$\forall 1 \leqslant n \leqslant N \, , \forall i \geqslant 0 \, , \theta_n^i \text{ is } \mathcal{F}_{n-1} - \text{ measurable}$$

So, θ is predictable, meaning the investor selects his portfolio of assets θ_n^i just before instant n, only knowing the information described by the filtration $\mathcal{F}_{n-1}$. Thus, he does note know the price S_n. The value of the portfolio at instant n is

$$V_n(\theta) = \theta_n \cdot S_n = \sum_{i=0}^{d} \theta_n^i S_n^i$$

which is $\mathcal{F}_n$-measurable.

Definition 28.2.1 *The strategy θ is self-financing if*

$$\theta_n \cdot S_n = \theta_{n+1} \cdot S_n \, , n \in \{1, ..., N-1\} \tag{28.2.4}$$

Thus, the quantity $\theta_n \cdot S_n$ is the portfolio value after the instant n and strictly before $n+1$. That is, θ_n is on $]n, n+1]$. The initial portfolio value is $V_0(\theta) = \theta_1 \cdot S_0$, which is $\mathcal{F}_0$-measurable. The gain process is given by

$$G_n(\theta) = \sum_{k=1}^{n} \theta_k \cdot \Delta S_k$$

where $\Delta S_k = S_k - S_{k-1}$. Assuming $G_0(\theta) = 0$, the self-financing condition can be written as

$$V_n(\theta) = \theta_1 \cdot S_0 + G_n(\theta) \, , n \in \{0, ..., N\}$$

Definition 28.2.2 *A strategy is admissible if*

1. *it is self-financing*
2. *$V_n(\theta) \geqslant 0$ for $n \in \{0, ..., N\}$*

Following such a strategy, the investor will never contract debts.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 28.2.3 *Two probabilities P and Q defined on the same probability space $(\Omega, \mathcal{F})$ are equivalent if for all $A \in \mathcal{F}$, $P(A) = 0 \Longleftrightarrow Q(A) = 0$.*

We are going to show that the no-arbitrage hypothesis is equivalent to the existence of a probability measure equivalent to P under which the discounted prices are martingales. We let $\overline{S}$ be the vector of discounted prices such that $\overline{S}_i(n) = \frac{S_i(n)}{S_0(n)}$. Then, we get the discounted portfolio

$$\overline{V}_n(\theta) = \frac{V_n(\theta)}{S_0(n)} = \sum_{i=0}^{d} \theta_n^i \cdot \overline{S}_i(n) = V_0(\theta) + \sum_{k=1}^{n} \theta_k \cdot \Delta \overline{S}_k$$

for $n > 0$ and $\overline{V}_0(\theta) = V_0(\theta)$. In that setting, the opportunities of arbitrage are the same for S and $\overline{S}$.

Lemma 28.2.1 *If there exist a martingale measure, that is, a probability Q equivalent to P such that the vector $\overline{S} = (1, \overline{S}_1, ..., \overline{S}_d)$ is a martingale under Q, then $\overline{V}_n(\theta)$ is a martingale under Q.*

Proof: We let $\overline{S}$ be a martingale under Q. One can prove this result by considering the self-financing strategy

$$\theta_n \cdot \overline{S}_n = \theta_{n+1} \cdot \overline{S}_n$$

where

$$\overline{V}_{n+1}(\theta) = \overline{V}_n(\theta) + \theta_{n+1} \cdot \Delta \overline{S}_{n+1}, n \in \{0, ..., N-1\} \tag{28.2.5}$$

We need to show that $\overline{V}_n(\theta)$ is a martingale under Q. Taking the expectation under the measure Q, we get

$$\begin{aligned} E^Q[\overline{V}_{n+1}(\theta) - \overline{V}_n(\theta)|\mathcal{F}_n] &= E^Q[\theta_{n+1} \cdot \Delta \overline{S}_{n+1}|\mathcal{F}_n] \\ &= E^Q[\theta_{n+1} \cdot (\overline{S}_{n+1} - \overline{S}_n)|\mathcal{F}_n] \\ &= \theta_{n+1} \cdot E^Q[(\overline{S}_{n+1} - \overline{S}_n)|\mathcal{F}_n] \end{aligned}$$

since θ_{n+1} is $\mathcal{F}_n$-measurable. Further, $\overline{S}_n$ is an $\mathcal{F}_n$-martingale under Q, so that

$$E^Q[(\overline{S}_{n+1} - \overline{S}_n)|\mathcal{F}_n] = 0$$

Theorem 28.2.1 *We assume that there is no opportunities of arbitrage. Then, there exists a probability Q equivalent to P, such that, under Q, the vector of discounted prices is a martingale.*

We let $\mathcal{P}$ be the set of all probabilities equivalent to P making the discounted prices a martingale.

Definition 28.2.4 *An $\mathcal{F}_N$ positive random variable X is replicable if there exists an admissible self-financing strategy θ such that $V_N(\theta) = X$.*

Assuming that this definition holds, the value $V_0(\theta)$ does not depend on the choice of θ. It is the value of X at instant 0. If $\mathcal{P}$ contains a unique measure Q, then $V_0(\theta) = E^Q[\overline{V}_N(\theta)]$.

Definition 28.2.5 *A price system is an application π from the set χ of $\mathcal{F}_n$-measurable positive random variables in $[0, \infty[$ such that*

1. *$\pi(X) = 0$ if and only if $X = 0$ P-p.s.*
2. *$\pi(\sum_{i=1}^{K} a_i X_i) = a_i \pi(X_i)$, $a_i \geqslant 0$, $X_i \in \chi$, $i \in \{1, ..., K\}$*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The system is coherent if

$$\pi[V_N(\theta)] = V_0(\theta)$$

for all admissible self-financing strategy θ. We denote $\prod$ the set of all coherent prices.

Definition 28.2.6 *A market is complete if all $\mathcal{F}_N$-measurable positive random variable X is replicable.*

That is, given an admissible self-financing strategy θ, we get $X = \sum_{i=0}^{d} \theta_N^i S_i(N)$. Thus,

$$\frac{X}{S_0(N)} = \overline{V}_0(\theta) + \sum_{n=1}^{N} \theta_n \cdot \Delta \overline{S}_n$$

We can rewrite the sum on the right-hand-side as $\int_0^N \theta d\overline{S}$, which is the integral of a step function equal to θ_n on $]n, n+1]$ with respect to $\overline{S}$.

***Proposition* 49** *A market is complete if and only if there exists a unique probability Q equivalent to P such that the discounted prices are martingales.*

Since $\overline{V}_n(\theta)$ is a Q-martingale, we get

$$V_n(\theta) = S_0(n) E^Q[\frac{X}{S_0(N)}|\mathcal{F}_n]$$

which is the price of X at instant n.

28.2.1.2 Example: Two assets model

We are now considering a market with two assets: a risk-free one and a risky one. The vector of price at the instant n is $S_n = (S_0(n), S_1(n))$ on the probability space $(\Omega, \mathcal{F}, P)$ with filtration $(\mathcal{F}_n)_{0 \leqslant n \leqslant N}$. For simplicity of exposition we let the risk-free asset satisfies $S_0(n) = 1$. We assume $S_1(n)$ is $\mathcal{F}_n$-measurable and belongs to $L_2(P)$. We denote $\Delta S_k = S_k - S_{k-1}$.

A self-financing strategy is a sequence of random variables (α_n, β_b) such that for all n, $0 \leqslant n \leqslant N$, we get

1. α_n takes values in $\mathbb{R}$, $\mathcal{F}_{n-1}$-measurable, $\alpha_0 = 0$ and $\sum_{j=1}^{n} \alpha_j \Delta S_j \in L_2(P)$.
2. β_n takes values in $\mathbb{R}$ and is adapted.
3. $\alpha_n S_1(n) + c \cdot \beta_n$ belongs to $L_2(P)$.

where α_n is the quantity invested in the risky asset. The value of the portfolio at instant $n \geqslant 0$ is

$$V_n(\alpha, \beta) = \alpha_n S_1(n) + \beta_n$$

which is $\mathcal{F}_n$-measurable. The gain at instant n of such a strategy is given by $G_n(\alpha) = \sum_{k=0}^{n} \alpha_k \Delta S_1(k)$. Following Follmer et al. [1986] and Schweizer [1988], we define for each strategy (α, β) a cost process $(C_n; n \geqslant 0)$ given by

$$C_n(\alpha, \beta) = V_n(\alpha, \beta) - G_n(\alpha)$$

Given Definition (28.2.1), a strategy is self-financing if the cost process is a constant denoted C_0. Then the value of the portfolio becomes

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$V_n(\alpha,\beta) = C_0 + G_n(\alpha)$$

In a self-financing strategy, the quantity β_n is $\mathcal{F}_{n-1}$-measurable, and, from Equation (28.2.4), we get

$$V_{n-1} = \alpha_n S_1(n-1) + \beta_n$$

so that $\beta_n = V_{n-1} - \alpha_n S_1(n-1)$. Replacing in the portfolio, we get the recursive equation

$$V_n(\alpha,\beta) = V_{n-1}(\alpha,\beta) + \alpha_n \Delta S_1(n) + C_n(\alpha,\beta) - C_{n-1}(\alpha,\beta) \tag{28.2.6}$$

where $\Delta S_1(n) = S_1(n) - S_1(n-1)$. In the case of an increasing cost process $C_n \geqslant C_{n-1}$, the investor must add money to refinance his portfolio.
In the case where the risk-free rate is non-null we can recover these results by taking the risk-free asset S_0 as numeraire. We obtain the vector of discounted prices $\overline{S}_n = (1, \overline{S}_1(n))$, where $\overline{S}_1(n) = \frac{S_1(n)}{S_0(n)}$, and the discounted portfolio

$$\overline{V}_n(\alpha,\beta) = \alpha_n \overline{S}_1(n) + \beta_n$$

In that setting, the gain at instant n is $\overline{G}_n(\alpha) = \sum_{k=0}^{n} \alpha_k \Delta \overline{S}_1(k)$ and the discounted cost process is

$$\overline{C}_n(\alpha,\beta) = \overline{V}_n(\alpha,\beta) - \overline{G}_n(\alpha)$$

The recursive relation for the discounted portfolio becomes

$$\overline{V}_n(\alpha,\beta) = \overline{V}_{n-1}(\alpha,\beta) + \alpha_n \Delta \overline{S}_1(n) + \overline{C}_n(\alpha,\beta) - \overline{C}_{n-1}(\alpha,\beta) \tag{28.2.7}$$

Thus, to simplify notation we consider the case where the risk-free rate is null so that $S_0(n) = 1$.

***Remark* 28.2.1** *Given Proposition (49), we can recover these results when $r \neq 0$ by taking the risk-free asset S_0 as numeraire and working under the Q-measure, where the discounted prices are martingales.*

28.2.1.3 Minimal risk in incomplete market

We are now going to define risk in incomplete market and minimise its effect. We are going to focus on the quadratic risk of the cost process introduced by Follmer et al. [1986] and Schweizer [1988]. Details are given in Appendix (28.2.4). For simplicity of exposition, we consider a market with two assets: a risk-free one and a risky one. Again, we let the risk-free asset satisfies $S_0(n) = 1$.

Definition 28.2.7 *We let Z be an $\mathcal{F}_N$-measurable positive random variable belonging to the Hilbert space $L_2(P)$. The strategy (α,β) generates Z if $V_N(\alpha,\beta) = Z$.*

The terminal cost associated to such a strategy is $C_N(\alpha,\beta) = Z - G_N(\alpha)$.
In a complete market there is a measure Q equivalent to P with one self-financing strategy replicating Z. In an incomplete market, for a given martingale measure, we need to find a strategy minimising the risk (which must be defined) in some sense.
We assume the existence of a probability Q such that, under Q, the price $S_1(n)$ is a square integrable martingale. Thus, the gain process is a Q-martingale with null expectation. Further, $E^Q[C_N]$ is equal to $E^Q[Z]$ and does not depend on the chosen strategy.
In a complete market Q is unique and we get a self-financing strategy α^*, β^* such that

$$Z = \alpha_N^* S_1(N) + \beta_N^* = G_N(\alpha) + E^Q[Z]$$

In that case $E^Q[C_N] = C_0$ defines the price of Z.
In an incomplete market we need to define the notion of risk associated to a strategy. We are going to focus on the quadratic risk of the cost process.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 28.2.8 *A generative strategy minimise the quadratic risk of cost if it minimises*

$$E^P[\big(C_N - E^P[Z]\big)^2]$$

The remaining risk at instant n, associated to strategy $\theta = (\alpha, \beta)$, is

$$R_n(\theta) = E^P[\big(C_N - C_n\big)^2|\mathcal{F}_n]$$

Definition 28.2.9 *A strategy $\tilde{\theta}$ is (n, θ)-coincident if $\tilde{\beta}_k = \beta_k$, $\forall k \leqslant n-1$, $\tilde{\alpha}_k = \alpha_k$, $\forall k \leqslant n$ and if it leads to the same terminal value than θ, that is $V_N(\tilde{\theta}) = V_N(\theta)$.*

Definition 28.2.10 *A strategy θ minimises the risk if, for all n and for all strategy $\tilde{\theta}$, (n, θ)-coincident, $R_n(\theta) \leqslant R_n(\tilde{\theta})$.*

We are now going to describe strategies minimising the quadratic risk under P. We let $\mathcal{G}$ be the set of random variables in $L_2(P)$ representing gains. They can be written as $\sum_{k=1}^N \alpha_k \Delta S_1(k)$. It is a closed vectorial sub-space of $L_2(P)$. Using orthogonal projection on $\mathcal{G}$, one can show that $Z - E^P[Z]$ can be uniquely decomposed as

$$Z - E^P[Z] = \sum_{k=1}^N \alpha_k^* \Delta S_1(k) + Z^* = G_N(\alpha^*) + Z^*$$

where Z^* is orthogonal to $\mathcal{G}$ and $E^P[Z^*] = 0$. Further, one can show that

$$\alpha_k^* = \frac{Cov_P(Z, \Delta S_1(k))}{Var_P(\Delta S_1(k))}$$

If the market is complete, we get $Z^* = 0$.

***Proposition* 50** *A strategy (α, β) generating Z is of minimal quadratic risk under P if and only if $\beta = \beta^*$ (as above). Then, we get*

$$E^P[\big(C_N(\alpha, \beta) - E^P[Z]\big)^2] = E^P[(Z^*)^2]$$

where $Z^* = Z - E^P[Z] - \sum_{k=1}^N \alpha_k^* \Delta S_1(k)$.

Here, the terminal value of β is given by $\beta_N = Z - \alpha_N S_1(N)$. There is a unique strategy θ^* minimising the risk

$$\theta^* = \big\{(\tilde{Z}_n - \alpha_n^* S_1(n), \alpha_n^*); 1 \leqslant n \leqslant N\big\}$$

where $\tilde{Z}_n = E^P[Z|\mathcal{F}_n]$.
We can also obtain the above results under the martingale measure Q. Note, we can not always find a strategy minimising the risk under P and we must consider a weaker notion of minimisation.

Definition 28.2.11 *Local risk*
Let θ be a strategy generating Z. We define

$$r_n(\theta) = E^P[\big(C_{n+1}(\theta) - C_n(\theta)\big)^2|\mathcal{F}_n]$$

Then, r_n is the local risk associated to θ.

***Proposition* 51** *There exists a strategy minimising the local risk.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Proof: We let E_n, Var_n and Cov_n be the conditional expectation under P with respect to $\mathcal{F}_n$, the conditional variance and conditional covariance, respectively. From the recursive relation in Equation (28.2.6), we can rewrite the local risk as

$$r_n(\theta) = E^P[\big(V_{n+1}(\theta) - \alpha_{n+1}\Delta S_1(n+1) - V_n(\theta)\big)^2|\mathcal{F}_n]$$

for all $n \in \{0, 1, ..., N-1\}$. Since the terminal condition is $V_N(\theta) = Z$ is given, we need to build α_{n+1} and $V_n(\theta)$ recursively. We denote $V_n = V_n(\theta)$. We get

$$\begin{aligned} r_n &= E_n[\big(V_{n+1}(\theta) - \alpha_{n+1}\Delta S_1(n+1) - V_n(\theta)\big)^2] \\ &= E_n[V_{n+1}^2] - E_n^2[V_{n+1}] + \big(E_n[V_{n+1} - \alpha_{n+1}\Delta S_1(n+1)] - V_n\big)^2 + \alpha_{n+1}^2\big(E_n[(\Delta S_1(n+1))^2] - E_n^2[\Delta S_1(n+1)]\big) \\ &- 2\alpha_{n+1}\big(E_n[V_{n+1}\Delta S_1(n+1)] - E_n[V_{n+1}]E_n[\Delta S_1(n+1)]\big) \end{aligned}$$

From the definition of variance and covariance,

$$\begin{aligned} Var(V_{n+1}) &= E_n[V_{n+1}^2] - E_n^2[V_{n+1}] \\ Var_n(\Delta S_1(n+1)) &= E_n[(\Delta S_1(n+1))^2] - E_n^2[\Delta S_1(n+1)] \\ Cov_n^2(\Delta S_1(n+1), V_{n+1}) &= E_n[V_{n+1}\Delta S_1(n+1)] - E_n[V_{n+1}]E_n[\Delta S_1(n+1)] \end{aligned}$$

we get

$$\begin{aligned} r_n &= Var(V_{n+1}) + \big(E_n[V_{n+1} - \alpha_{n+1}\Delta S_1(n+1)] - V_n\big)^2 + \alpha_{n+1}^2 Var_n(\Delta S_1(n+1)) - 2\alpha_{n+1}Cov_n(\Delta S_1(n+1), V_{n+1}) \\ &= Var(V_{n+1}) + \big(E_n[V_{n+1} - \alpha_{n+1}\Delta S_1(n+1)] - V_n\big)^2 + Var_n(\Delta S_1(n+1))\big(\alpha_{n+1} - \frac{Cov_n(\Delta S_1(n+1), V_{n+1})}{Var_n(\Delta S_1(n+1))}\big)^2 \\ &- \frac{Cov_n^2(\Delta S_1(n+1), V_{n+1})}{Var_n(\Delta S_1(n+1))} \end{aligned}$$

As a result, the minimising strategy is given by

$$\begin{cases} V_n(\theta^*) = E_n[V_{n+1}(\theta^*) - \alpha_{n+1}^*\Delta S_1(n+1)] \\ \alpha_{n+1}^* = \frac{Cov_n(\Delta S_1(n+1), V_{n+1})}{Var_n(\Delta S_1(n+1))} \end{cases} \tag{28.2.8}$$

Note that

$$E_n[\Delta C_{n+1}(\theta^*)] = 0$$

so that the cost process is a martingale.

28.2.2 The continuous model

28.2.2.1 The stochastic integral

We let the space $(\Omega, \mathcal{F})$ contains the tribes $(\mathcal{F}_t, t \in \mathbb{R}_+)$ which describe information. If $A \in \mathcal{F}_t$, the agent knows, at time t, if A is occurred or not. We let $\mathcal{F}_t$ be the filtration associated to the Brownian motion, that is, $\mathcal{F}_t = \sigma(W_s, s \leqslant t)$. This filtration is increasing, that is, $\mathcal{F}_s \subset F_t$ for $s \leqslant t$. The process X_t is $\mathcal{F}_t$-adapted if X_t is $\mathcal{F}_t$-measurable for all t. We let the stock price $S_t = W_t$ be a Brownian motion and assume we hold $\theta(t)$ stock at time t. We consider a self-financing portfolio with initial value x, and trading occurs at instants t_k. The wealth at time t_K is given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$x + \sum_{k=1}^{K} \theta(t_k)\big(W_{t_k} - W_{t_{k-1}}\big)$$

In order to trade at all time t we must consider the stochastic integral $\int_0^T \theta(s)dB(s)$. In the discrete case (see Appendix (28.2.1)), we have assumed that $\theta(t_k)$ was $\mathcal{F}_{t_{k-1}}$-measurable, called a predictable process. In continuous time, a continuous left process is predictable. We also need the following integrability condition on θ: Given the set of predictable processes Θ such that

$$\int_0^T \theta^2(s)ds < \infty$$

For all $\theta \in \Theta$, the stochastic integral $\int_0^T \theta(s)dW(s)$ is well defined. Note, $\int_s^t dW(u) = W_t - W_s$. The stochastic integral with respect to the Brownian motion is a martingale.

Proposition 52 *If $\theta \in \Theta$ and $E[\int_0^T \theta^2(s)ds] < \infty$, the stochastic integral $M_t = \int_0^t \theta(s)dW_s$, $t \leqslant T$, defines a process M which is a martingale with null expectation and variance $E[M_t^2] = \int_0^t \theta^2(s)ds$.*

28.2.2.2 Description

We let T be the market horizon, and assume $(d+1)$ assets $S = (S_0, S_1, .., S_d)$ traded between 0 and T. We let $S_i(t)$ be the price of the ith asset at time t with price process continuous in time. The asset S_0 is the cash describing the value of \$1 daily compounded at the bank. It is a risk-free asset earning rdt in the time interval $[t, t+dt]$ known at time t. The dynamics of the continuous time model, under the historical measure P, are

$$\begin{aligned} dS_0(t) &= S_0(t)r(t)dt\ ,\ S_0(0) = 1 \\ dS_i(t) &= S_i(t)\big(b_i(t)dt + \sum_{j=1}^{d} \sigma_{ij}d\widehat{W}_j(t)\big) \end{aligned}$$

where $\widehat{W}$ is a d-dimensional Brownian motion. The processes r, b, σ are measurable, $\mathcal{F}_t$-adapted and bounded. $\mathcal{F}_t$ is the tribe composed of the trajectories of W up to t, $r > 0$ and σ is invertible. In that setting the risk-free asset is given by $S_0(t) = e^{\int_0^t r(s)ds}$.
Assuming complete market, the agent has an initial capital x at time 0 and want to maximise the expected value of a utility function on consumption and final wealth. As a constraint he can not have debts, that is, he can not borrow (requires $X \geqslant 0$). The agent builds a portfolio with $\theta_i(t)$, $i = 0, 1, ..., d$, assets at time t, which can be positive or negative.
A portfolio strategy is given by the process $(\theta_i(t))_{0 \leqslant i \leqslant d}$ corresponding to the quantity invested in each asset. We consider simple strategies where the composition of the portfolio changes on a finite set of dates called trading dates. In discrete time, any strategy is a simple strategy.

Definition 28.2.12 *A simple strategy written on all assets is given by a finite set of trading dates*

$$\mathcal{T} = \{(t_i)_{0 \leqslant i \leqslant n}; 0 = t_0 < t_1 < t_2 < ... < t_n = T\}$$

and $(d+1)$ processes $(\theta_i(t))_{0 \leqslant i \leqslant d}$ giving the allocation of the assets in the portfolio over time

$$\theta_i(t) = n_0^i I_{[0,t_1]}(t) + \cdots + n_k^i I_{]t_k,t_{k+1}]}(t) + \cdots + n_{N-1}^i I_{]t_{N-1},t_N]}(t)$$

where the variables n_k^i are $\mathcal{F}_{t_k}$-measurable.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

His wealth (or portfolio value) at time t is

$$V_t(\theta) = X(t) =< \theta(t), S_t >= \sum_{i=0}^{d} \theta_i(t) S_i(t)$$

***Remark* 28.2.2** *For all t in the interval $]t_k, t_{k+1}]$ then $\theta_i(t) = \theta_i(t_{k+1}) = n^i(k)$. That is, the part invested in the ith asset is $\mathcal{F}_{t_k}$-measurable and only depends on the information available at the previous trading date. Hence, the process $\theta_i(t)$ is predictable and the process $X(t)$ is adapted. In continuous time, as the simple strategies are processes left-continuous, so is the value of a simple portfolio.*

28.2.2.3 The self-financing condition

28.2.2.3.1 Description Between the dates t_k and t_{k+1} an investor following the strategy θ allocates n_k^i units in asset S_i. Just before the trading date t_{k+1}, the portfolio value is $< n_k, S_{t_{k+1}} >$. At time t_{k+1}, the investor build a new portfolio, that is, reallocate the weights of the portfolio from the information at time t_{k+1}. Assuming no cash is added or removed from the portfolio at time t_{k+1}, then the self-financing condition can be expressed as

$$< n_k, S_{t_{k+1}} >=< n_{k+1}, S_{t_{k+1}} > \tag{28.2.9}$$

When considering the assets' variation between two trading dates, the self-financing condition becomes

$$V_{t_k}(\theta)+ < n_k, S_{t_{k+1}} - S_{t_k} >=< n_k, S_{t_{k+1}} >= V_{t_{k+1}}(\theta) \tag{28.2.10}$$

***Remark* 28.2.3** *The variations of a self-financing portfolio are only due to the variations of the assets.*

To see this, we start from the definition of the portfolio, where we have $V_{t_k}(\theta) =< \theta(t_k), S_{t_k} >$ and $V_{t_{k+1}}(\theta) =< \theta(t_{k+1}), S_{t_{k+1}} >$. Taking their difference, and using Remark (28.2.2), we get

$$\begin{aligned} V_{t_{k+1}}(\theta) - V_{t_k}(\theta) &= < \theta(t_{k+1}), S_{t_{k+1}} - S_{t_k} > \\ &= < n_k, S_{t_{k+1}} - S_{t_k} > \end{aligned}$$

More generally, given the portfolio $V_t(\theta)$, its variation satisfies

$$V_t(\theta) - V_0(\theta) = \int_0^t < \theta(u), dS_u > \ , \forall t \leqslant T$$

which is the gain process. We can prove that it is a self-financing condition in the discrete case (see Appendix 28.2.1) by letting S_t be constant on $[t_k, t_{k+1}[$ and θ be constant on $]t_k, t_{k+1}]$. Then,

$$\begin{aligned} < \theta(k+1), S_{k+1} > &= < \theta(k), S_k > + \int_k^{k+1} \theta(u) dS_u \\ &= < \theta(k), S_k > + < \theta(k+1), (S_{k+1} - S_k) > \end{aligned}$$

such that

$$< \theta(k+1), S_k >=< \theta(k), S_k >$$

which corresponds to Definition (28.2.1).
A self-financing portfolio is a strategy to buy or sell assets whose value is not modified by adding or removing cash. Further, the self-financing condition implies that the value of the portfolio does not jump at trading dates. The self-financing condition is a necessary and sufficient condition for the continuity process of the value of the portfolio.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

28.2.2.3.2 Discounted prices If we choose the asset S_0 as numeraire, then the stochastic integral

$$\frac{x}{S_0(0)} + \int_0^t < \theta(u), d(\frac{S(u)}{S_0(u)}) >$$

is the value of a self-financing portfolio.
We let N_t be the dollar value of a numeraire. Then, $\frac{S_i(t)}{N_t}$ is the price of the ith asset in that numeraire and the value of the portfolio is $\frac{V_t(\theta)}{N_t}$. If θ is a simple strategy satisfying the self-financing condition in Equation (28.2.9) in the initial numeraire, then it satisfies the same equation in the new numeraire:

$$< n_k, \frac{S_{t_{k+1}}}{N_{t_{k+1}}} >=< n_{k+1}, \frac{S_{t_{k+1}}}{N_{t_{k+1}}} > \tag{28.2.11}$$

In the case of a self-financing portfolio $V_t(\theta) =< \theta_t, S_t >$ where $dV_t(\theta) =< \theta_t, dS_t >$, we get

$$\frac{V_t(\theta)}{N_t} =< \theta_t, \frac{S_t}{N_t} > \text{ and } d\frac{V_t(\theta)}{N_t} =< \theta_t, d\frac{S_t}{N_t} >$$

28.2.2.4 Valuation and dynamics

***Hypotheses* 1** *There exists a probability measure Q equivalent to P such that the vector of discounted prices $\frac{S_t}{S_0(t)}$ is a Q-martingale.*

Definition 28.2.13 *A martingale measure Q is a probability measure equivalent to P such that, under Q, discounted prices are martingales.*

Definition 28.2.14 *Assuming no-arbitrage opportunities, if θ is a strategy financing Z, $\theta_0 \cdot S_0$ is the arbitrage price of Z and $\pi(Z)_t = \theta_t \cdot S_t$ is the implicit price of Z at time t.*

***Proposition* 53** *Assuming no-arbitrage opportunities, the arbitrage price is well defined.*

Given the dynamics of our model and assuming θ is an adapted vectorial process, we can infer the dynamics of the portfolio as follows:

$$\begin{aligned} dV_t(\theta) &= \sum_{i=0}^{d} \theta_i(t) dS_i(t) \\ &= \theta_0(t)S_0(t)rdt + \sum_{i=1}^{d} \theta_i(t)S_i(t)b_i(t)dt + \sum_{i=1}^{d}\sum_{j=1}^{k} \theta_i(t)S_i(t)\sigma_{ij}(t)d\widehat{W}_j(t) \\ &= \theta_0(t)S_0(t)rdt + < (\theta S)_t, b_t > dt + < (\theta S)_t, \sigma_t d\widehat{W}(t) > \end{aligned}$$

where $(\theta S)_t$ is a vector with components $(\theta_i(t)S_i(t); i = 1, .., d)$. Using the self-financing equation, the dynamics of the portfolio simplify to

$$dV_t = V_t rdt + < (\theta S)_t, b_t - r1 > dt + < (\theta S)_t, \sigma_t d\widehat{W}(t) > \tag{28.2.12}$$

where 1 is a vector with all components equal to one. Note, a process satisfying this equation is a self-financing portfolio corresponding to investing $(\theta S)_t$ in risky assets and $\frac{1}{S_0(t)}\left(V_t(\theta) - \sum_{i=1}^{d} \theta_i(t)S_i(t)\right)$ in a risk-free asset.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

28.2.2.5 Some examples

As an example, we consider a portfolio made of two assets: a risk-free one S_0 and a risky one S_1. That is, $S_t = (S_0(t), S_1(t))$.
In the first strategy we present, part of the portfolio is invested in stocks and the rest is put at the bank and compounded with the gross rate in Equation (22.1.1). Here, the portfolio is not rebalanced before maturity T. At time 0, the wealth x is invested into n^1 stocks such that $x = n^1 S_1(0) + (x - n^1 S_1(0))$. At all time t the value of the portfolio is

$$V(t) = n^1 S_1(t) + (x - n^1 S_1(0))e^{\int_0^t r_s ds}$$

with dynamics

$$\begin{aligned} dV(t) &= n^1 dS_1(t) + (x - n^1 S_1(0))e^{\int_0^t r_s ds} r_t dt \\ &= n^1 dS_1(t) + (V(t) - n^1 S_1(t)) r_t dt \end{aligned}$$

We let $P(t,T) = E[e^{-\int_t^r r_s} | \mathcal{F}_t]$ be the price of a zero-coupon bond. We also let $C(t,T) = P^{-1}(t,T)$ be the compounding factor. We can assume that the quantity $(x - n^1 S_1(0))$ is invested at the bank until maturity T. The value at time t of \$1 invested in 0 till maturity T is $C(0,T)P(t,T) = C(0,t)$. Thus, the value of the portfolio at time t is

$$V(t) = n^1 S_1(t) + (x - n^1 S_1(0))C(0,t)$$

Another strategy is to let θ be represented by the pair (θ_0, θ_1) and allow for rebalancing at all trading time. Then,

$$\begin{aligned} < \theta_t, S_t > &= \theta_0(t)S_0(t) + \theta_1(t)S_1(t) \\ &= < \theta_0, S_0 > + \int_0^t \theta_u dS_u \\ &= \theta_0(0)S_0(0) + \theta_1(0)S_1(0) + \int_0^t \theta_0(u)dS_0(u) + \int_0^t \theta_1(u)dS_1(u) \end{aligned}$$

In our example, the self-financing condition, defined in Equation (28.2.9), becomes

$$n_k^0 S_0(t_{k+1}) + n_k^1 S_1(t_{k+1}) = n_{k+1}^0 S_0(t_{k+1}) + n_{k+1}^1 S_1(t_{k+1})$$

28.2.3 The BS-formula

We use the replication argument and the self-financing condition in the continuous time to derive the pricing equations. This technique was presented by Black et al. [1973] to derive the partial differential equation governing the option price. The method was further developed by Harrison et al. [1979] [1981], leading to the martingale pricing technique.

28.2.3.1 The historical measure

We consider a market with two assets

$$dS_0(t) = rS_0(t)dt \text{ and } dS_1(t) = \mu S_1(t)dt + \sigma S_1(t)d\widehat{W}_t$$

where $\widehat{W}_t$ is a Brownian motion under the historical measure P. There exists a probability Q equivalent to P such that under Q the discounted prices are martingales. We consider the random variable $Z = g(S_1(T))$, which is positive $\mathcal{F}_T$-measurable. There exists $C \in C^{1,2}([0,T] \times \mathbb{R}_+, \mathbb{R})$ such that

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\pi(Z)_t = C(t, S_1(t)) \text{ , } t < T \text{ and } g(x) = C(T, x) \text{ , } x \in \mathbb{R}_+$$

where $\pi(\cdot)_t$ is the implicit price given in Definition (28.2.14).
We let $Y_t = C(t, S_1(t))$ and apply Ito's lemma, getting

$$dY_t = \mathcal{L}C(t, S_1(t))dt + \sigma S_1(t)C_x(t, S_1(t))d\widehat{W}_t$$

where

$$\mathcal{L}C = \mu x C_x + C_t + \frac{1}{2}\sigma^2 x^2 C_{xx}$$

We assume there is a strategy θ financing Z given by the pair (α, β). We get $S_t = (S_0(t), S_1(t))$ and

$$\begin{aligned}
\theta_t \cdot S_t &= \alpha_t S_1(t) + \beta_t S_0(t) \\
&= \theta_0 \cdot S_0 + \int_0^t \theta_u dS_u \\
&= \alpha_0 S_1(0) + \beta_0 S_0(0) + \int_0^t \alpha_u dS_1(u) + \int_0^t \beta_u dS_0(u) \\
&= C(t, S_1(t)) = Y_t
\end{aligned}$$

Thus, we can also write dY_t as follows

$$\begin{aligned}
dY_t &= \alpha_t dS_1(t) + \beta_t dS_0(t) \\
&= \alpha_t\big(\mu S_1(t)dt + \sigma S_1(t)d\widehat{W}_t\big) + \beta_t r S_0(t)dt \\
&= \big(\beta_t r S_0(t) + \mu\alpha_t S_1(t)\big)dt + \sigma\alpha_t S_1(t)d\widehat{W}_t
\end{aligned}$$

We now have two Ito expressions for the variation dY_t. Using the Unique Decomposition Theorem (see Duffie [1992]), the drift and diffusion terms must be equal. Comparing the two SDE and equating terms in dt, we get

$$\mathcal{L}C(t, S_1(t)) = \beta_t r S_0(t) + \mu\alpha_t S_1(t)$$

and equating terms in $d\widehat{W}_t$, we get

$$\sigma S_1(t)C_x(t, S_1(t)) = \sigma\alpha_t S_1(t)$$

From this equation, we get

$$\alpha_t = C_x(t, S_1(t))$$

Since

$$\alpha_t S_1(t) + \beta_t S_0(t) = C(t, S_1(t))$$

then we can deduce

$$\beta_t = \frac{1}{S_0(t)}\big(C(t, S_1(t)) - \alpha_t S_1(t)\big)$$

Plugging this back in the terms in dt, we get the partial differential equation (PDE)

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\mathcal{L}C(t, S_1(t)) = r\big(C(t, S_1(t)) - S_1(t)C_x(t, S_1(t))\big) + \mu C_x(t, S_1(t))S_1(t)$$

Replacing $\mathcal{L}C(t, S_1(t))$ with its value and simplifying, we get

$$rS_1(t)C(t, S_1(t)) + C_t(t, S_1(t)) + \frac{1}{2}\sigma^2 S_1^2(t)C_{xx}(t, S_1(t)) = rC(t, S_1(t))\,,\, t \in [0, T]$$

with terminal condition $C(T, S_1(T)) = g(S_1(T))$.

***Remark* 28.2.4** *μ does not appear in the above parabolic equation.*

This equation is still satisfied if we replace $S_1(t)$ by x, $x > 0$. We get the following theorem:

Theorem 28.2.2 *We let $S_0(t)$ have dynamics $dS_0(t) = rS_0(t)dt$ and the risky asset $S_1(t)$ have the dynamics $dS_1(t) = \mu S_1(t)dt + \sigma S_1(t)d\widehat{W}_t$. We let $Z = g(S_1(T))$ be a positive random variable and $\pi(Z)_t$ be its implicit price. There exists $C \in C^{1,2}([0, T] \times \mathbb{R}_+, \mathbb{R})$ such that*

$$\pi(Z)_t = C(t, S_1(t))\,,\, t < T \text{ and } g(x) = C(T, x)\,,\, x \in \mathbb{R}_+$$

Then, C satisfies the above parabolic equation

$$rxC(t, x) + C_t(t, x) + \frac{1}{2}\sigma^2 x^2 C_{xx}(t, x) = rC(t, x)\,,\, x \in]0, \infty[\,,\, t \in]0, T[\tag{28.2.13}$$

with terminal condition $C(T, x) = g(x)$, $x \in]0, \infty[$. The strategy θ financing Z is given by the pair (α, β) where

$$\alpha_t = C_x(t, S_1(t)) \text{ and } \beta_t = \frac{1}{S_0(t)}\big(C(t, S_1(t)) - \alpha_t S_1(t)\big)$$

Given x and t fixed, we let the process $Z_{x,t}(s)$, $t \leqslant s \leqslant T$ be given by

$$Z_{x,t}(s) = x + r\int_t^s Z_{x,t}(u)du + \sigma\int_t^s Z_{x,t}(u)d\widehat{W}_u$$

where $Z_{x,t}(t) = x$ After the time t, it follows the same dynamic as

$$\frac{dZ_u}{Z_u} = rdu + \sigma d\widehat{W}_u$$

We get the Feymann-Kac formula:

Theorem 28.2.3 *If $g \in C^2(\mathbb{R})$ is in positive values and $g, g^{'}, g^{''}$ lipschitz, then the function*

$$C(t, x) = E[e^{-r(T-t)}g(Z_{x,t}(T))]$$

is the unique solution to the parabolic equation (28.2.13).

The price of the option is the initial investment in the replicating portfolio.

***Remark* 28.2.5** *This approach is not suitable for non-Markovian settings, such as stochastic volatility in the underlying asset or path-dependent options.*

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

28.2.3.2 The risk-neutral measure

We now follow the martingale approach to derive the pricing equations. We let L_t be a process satisfying $dL_t = -\frac{(\mu-r)}{\sigma}L_t d\widehat{W}_t$, with $L_0 = 1$. Using Girsanov's theorem, the process $\overline{S}_1(t)$ satisfies $d\overline{S}_1(t) = \overline{S}_1(t)dW_t$ where $W_t = \widehat{W}_t + \frac{(\mu-r)}{\sigma}t$ is a Q-Brownian motion. The probability Q is defined by $dQ = L_T dP$ and is equivalent to P. The dynamics of the risky asset, under Q, are

$$\frac{dS_1(t)}{S_1(t)} = rdt + \sigma dW_t$$

In that setting, the portfolio

$$Y_t = \alpha_t S_1(t) + \beta_t S_0(t)$$

satisfies the dynamics

$$\begin{aligned} dY_t &= \alpha_t S_1(t)\big(rdt + \sigma dW_t\big) + \beta_t S_0(t) rdt \\ &= Y_t rdt + dM_t \end{aligned}$$

where

$$dM_t = \alpha_t S_1(t)\sigma dW_t$$

Assuming some integrability condition, M_t is a martingale under Q. As a result, $\overline{Y}_t = e^{-rt}Y_t$ is a martingale and we have

$$e^{-rt}Y_t = E^Q[e^{-rT}Y_T|\mathcal{F}_t]$$

Alternatively, since the process S_t is a Markov process, we get

$$Y_t = C(t, S_1(t)) = E^Q[e^{-r(T-t)}g(S_1(T))|\mathcal{F}_t]$$

and

$$d\overline{Y}_t = \gamma_t dW_t$$

for some process γ. We let $\Gamma = \frac{e^{rt}\gamma}{Y}$ and apply Ito's formula, getting

$$\frac{dY_t}{Y_t} = rdt + \Gamma_t dW_t$$

We can choose

$$\alpha = \frac{\Gamma Y}{\sigma S}$$

and see that both dynamics have the same evolution for all time t.

Theorem 28.2.4 *The implicit price of $g(S_1(T))$ is given by*

$$E^Q[e^{-r(T-t)}g(S_1(T))|\mathcal{F}_t]$$

where Q is the risk-neutral probability.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

***Remark* 28.2.6** *Note, for $t = 0$, we have*

$$C(0,x) = E^Q[e^{-rT}g(S_1(T))] = E^P[e^{-rT}g(Z_T)]$$

where Z_t satisfies $\frac{dZ_t}{Z_t} = rdt + \sigma d\widehat{W}_t$, $Z_0 = x$, and we recover the PDE in Equation (28.2.13).

We can show that we recover the PDE in Equation (28.2.13) for any t by using

$$S_1(T) = S_1(t)e^{r(T-t)-\frac{1}{2}\sigma^2(T-t)-\sigma(W_T-W_t)}$$

Applying Ito's formula to $C(t, S_1(t))$, we get

$$dC(t, S_1(t)) = C_t(t, S_1(t))dt + C_x(t, S_1(t))dS_1(t) + \frac{1}{2}S_1^2(t)\sigma^2 C_{xx}(t, S_1(t))dt$$

Since $e^{-rt}C(t, S_1(t))$ is a martingale, we can set the terms in dW_t to zero, and we recover Equation (28.2.13). Thus, we can decompose $C(t, S_1(t))$ as

$$C(t, S_1(t)) = \alpha_t S_1(t) + \beta_t S_0(t)$$

where

$$\alpha_t = C_x(t, S_1(t)) \text{ and } \beta_t = \frac{1}{S_0(t)}\big(C(t, S_1(t)) - C_x(t, S_1(t))S_1(t)\big)$$

Recall, a process satisfying $\frac{dZ_s}{Z_s} = rds + \sigma d\widehat{W}_t$ has a log-normal law. In our setting the initial time is t and $Z_{x,t}(s)$ follows the law $N\big(\log Z_{x,t}(t) + (r - \frac{1}{2}\sigma^2)(s-t), \sigma^2(s-t)\big)$. We can then compute

$$e^{-r(T-t)}E[g(Z_{x,t}(T))] = e^{-r(T-t)}\int_{-\infty}^{\infty} g(e^u)f_{T-t}(u)du$$

where $f_{T-t}(u)$ is the density function of a normal law $N(\log x + (r - \frac{1}{2}\sigma^2)(T-t), \sigma^2(T-t))$. In the case of a call option we have $g(x) = (x-K)^+$. Replacing in the integral we get a closed-form solution.

28.2.3.3 The dynamics of the price

We start with the self-financing condition in Equation (28.2.9) and write the portfolio variation between two dates as

$$V_{t_{k+1}}(\theta) = V_{t_k}(\theta) + < n_k, S_{t_{k+1}} - S_{t_k} >$$

In a market with two assets, we get

$$V_{t_{k+1}}(\theta) = V_{t_k}(\theta) + \alpha_k\Delta_k S_1 + \beta_k\Delta_k S_0$$

where $\Delta_k S_i = S_i(t_{k+1}) - S_i(t_k)$, $i = 0, 1$. Recall, in the BS-framework $V_{t_k}(\theta) = C(t_k, S_1(t_k))$ is the option price. Given the dynamics of the underlying assets, under the historical measure P, we get

$$\Delta_k S_0 = S_0(t_k)\big(e^{r\Delta t} - 1\big) \text{ and } \Delta_k S_1 = S_1(t_k)\big(e^{(\mu-\frac{1}{2}\sigma^2)\Delta t + \sigma(\widehat{W}_{t_{k+1}} - \widehat{W}_{t_k})} - 1\big)$$

where $\Delta t = t_{k+1} - t_k$. We let $\Delta V_{t_k} = V_{t_{k+1}}(\theta) - V_{t_k}(\theta)$ be the portfolio variation. Given the values of α and β in Theorem (28.2.2), we can rewrite the latter as

$$\begin{aligned}\Delta V_{t_k} &= C_x(t_k, S_1(t_k))S_1(t_k)\big(e^{(\mu-\frac{1}{2}\sigma^2)\Delta t + \sigma(\widehat{W}_{t_{k+1}} - \widehat{W}_{t_k})} - 1\big) \\ &+ \big(C(t_k, S_1(t_k)) - C_x(t_k, S_1(t_k))S_1(t_k)\big)\big(e^{r\Delta t} - 1\big)\end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Lemma 28.2.2 *The Laplace transform of a Gaussian random variable X with mean μ and variance σ^2 is given by*

$$E[e^{\lambda X}] = e^{\lambda\mu + \frac{1}{2}\sigma^2\lambda^2}$$

Taking the conditional expectation with respect to time t_k, we get

$$E_{t_k}[\Delta V_{t_k}] = C(t_k, S_1(t_k))\big(e^{r\Delta t} - 1\big) + C_x(t_k, S_1(t_k))S_1(t_k)\Big[\big(e^{\mu\Delta t} - 1\big) - \big(e^{r\Delta t} - 1\big)\Big]$$

Taking the limit $\Delta t \to 0$ and accounting for the Ito's term, the dynamics of the portfolio variation become

$$dV_{t_k} = C(t_k, S_1(t_k))rdt + C_x(t_k, S_1(t_k))S_1(t_k)\big((\mu - r)dt + \sigma d\widehat{W}_{t_k}\big)$$

We let $W_t = \widehat{W}_t + \frac{(\mu - r)}{\sigma}t$ be a Q-Brownian motion and rewrite the above dynamics as

$$dV_{t_k} = C(t_k, S_1(t_k))rdt + C_x(t_k, S_1(t_k))S_1(t_k)\sigma dW_{t_k}$$

We recover the dynamics in Equation (28.2.12), where the portfolio of risky assets (vector describing the amount to be invested in each stock) is given by

$$\pi_t = (\theta S)_t = C_x(t, S_1(t))S_1(t) \tag{28.2.14}$$

Note, the discounted portfolio value $\overline{V}$ is a Q-martingale. This is the dynamics of a European option in the BS-framework.

28.2.4 The replicating portfolio

In a complete market, every contingent claim can be replicated with a trading strategy, or replicating portfolio, made of the underlying asset X and a risk-free money account. In this sense, the claim is redundant. For instance, Black and Scholes derived the price of an option by constructing a replicating portfolio (see details in Section (22.1.1.2)). This is not the case in an incomplete market, where there exists non-redundant claims carrying an intrinsic risk.

In the absence of arbitrage opportunities, there is an equivalent probability measure $Q \approx P$ such that the stochastic process $X = \{X_t; 0 \leqslant t \leqslant T\}$, on some probability space $(\Omega, \mathcal{F}, \mathbb{P})$, is a martingale under P^*. It implies that X is a semimartingale under the historical measure P. Completeness means that any claim H can be represented as a stochastic integral of X. In the incomplete case, the claim is not necessarily a stochastic integral of X. The claim has an intrinsic risk. Any portfolio strategy generating such a claim will involve a random process of cumulative costs. Thus, we need to construct strategies minimising this risk. In continuous-time, risk-minimisation is defined in a local sense, and the construction of such strategies is reduced to a stochastic optimality equation with solution equivalent to a projection problem in the space $\mathcal{S}^2$ of semimartingales.

We are briefly going to introduce the theory of portfolio replication in complete and incomplete markets (details can be found in Follmer et al. [1990], Oksendal [1998]).

We let $(X(t))_{t\in[0,T]}$ be a continuous semimartingale on the horizon $[0, T]$, representing the price process of a risky asset. The Doob-Meyer decomposition of X is

$$X = X_0 + M + A$$

where $M = (M(t))_{t\in[0,T]}$ is a (continuous) local martingale and $A = (A(t))_{t\in[0,T]}$ is a predictable process with paths of bounded variation. It amounts to the integrability condition

$$E[X_0^2 + < X >_T + |A|_T^2] < \infty$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

where $< X >=< M >$ is the quadratic variation of X and $|A|$ is the total variation of A. Note, this setting does not account for jumps, and X is a predictable process and not an optional process (see Appendix (25.1.1.1)).
We let a contingent claim be given by $H = h(X(T))$ for some function h, where H is a random variable such that $H \in \mathcal{L}^2(\Omega, \mathcal{F}, \mathbb{P})$. We let $M(t)$ be the risk-free money account, $\alpha(t)$ is the amount of asset held at time t, and $\beta(t)$ is the money account held at time t. A trading strategy φ is a pair of processes $(\alpha(t), \beta(t))$ where α is predictable and β is adapted.

28.2.4.1 Complete market

In a complete market, the value of a portfolio at time t satisfies

$$V(t) = \alpha(t)X(t) + \beta(t)M(t)\,, 0 \leqslant t \leqslant T \tag{28.2.15}$$

For simplicity of exposition we set $M(t) = 1$ for all $0 \leqslant t \leqslant T$. The trading strategy (α, β) is admissible, such that the value process $V(t)$ is square-integrable and have right-continuous paths defined by

$$V(t) = V_0 + \int_0^t \alpha(s)dX(s)$$

For $\mathbb{Q}$-almost surely, every contingent claim H is attainable and admits the representation

$$V(T) = H = V_0 + \int_0^T \alpha(s)dX(s)$$

where $V_0 = E^Q[H]$.
The strategy is self-financing [1], meaning the accumulated cost of the portfolio up to time t is a constant

$$C_t = V(t) - \int_0^t \alpha(s)dX(s) = V_0$$

where V_0 is a perfect hedge. $C(\varphi)$ is the (cumulative) cost process of φ.
Derivation of the solution: We admit strategies (α, β) such that V and C are square integrable, with right-continuous paths and satisfying

$$V(T) = H\,, P- \text{ a.s.}$$

We assume that the claim H admits an Ito representation of the form

$$H = H_0 + \int_0^T \alpha^H(s)dX(s)\,, P- \text{ a.s.}$$

where α^H satisfies some integrability condition. Then, we can use the strategy defined by

$$\alpha = \alpha^H\,, \beta = V - \alpha \cdot X\,, V(t) = H_0 + \int_0^t \alpha^H(s)dX(s)$$

This strategy is admissible and self-financing, that is,

$$C_t = C_T = H_0 \text{ for } 0 \leqslant t \leqslant T$$

Thus, the above Ito representation leads to a strategy producing the claim H from the amount $C_0 = H_0$. No risk is involved.

[1] All future changes in the hedge portfolio are funded from an initially set bank account, without any cash injection or withdrawal over the option lifetime.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We want to exclude arbitrage opportunities and consider the equivalent martingale measure Q. We assume that $Q \approx P$ is a probability measure on $(\Omega, \mathbb{P})$ such that

$$\frac{dQ}{dP} \in L_2(\Omega, \mathcal{F}, \mathbb{P})$$

and X is a martingale under Q. Then the above strategy can be identified as

$$V(t) = E^Q[H|\mathcal{F}_t] \, , 0 \leqslant t \leqslant T$$

and α^H is obtained as the Radon-Nikodym derivative

$$\alpha^H = \frac{d < V, X >}{d < X >} \tag{28.2.16}$$

where $< V, X >$ is the covariance process associated to V and X. Thus, the strategy can be identified in terms of Q and does not depend on the choice of the initial measure $P \approx Q$.

28.2.4.2 Incomplete market

Exposures to uncertain future events constitutes a basis of arbitrage opportunity, called intrinsic risk. Since the complete market theory failed to account for well-known market anomalies, an incomplete market theory developed to understand and explain such anomalies. In order to reduce this risk, Follmer et al. [1990] derived a replicating portfolio in incomplete market.
We consider the case $P = Q$ where X is already a martingale under the initial measure P. Follmer et al. [1986] introduced the criterion of risk-minimisation: we look for an admissible strategy minimising, at each time t, the remaining risk

$$R_t(\varphi) = E[\big(C_T - C_t\big)^2|\mathcal{F}_t] \, , 0 \leqslant t \leqslant T \tag{28.2.17}$$

over all admissible continuation of this strategy from time t on. R_t is a conditional mean-square error process. H is attainable if and only if this remaining risk can be reduced to zero.
However, for a general claim in L_2, the cost process associated with a risk-minimising strategy is no-longer self-financing. It is mean-self-financing in the sense that

$$E[C_T - C_t|\mathcal{F}_t] = 0 \, , 0 \leqslant t \leqslant T$$

That is, the cost process $C(\varphi)$ is a martingale.

***Remark* 28.2.7** *The R-minimality R_t is the conditional variance of the total cost $C_T(\varphi)$, given the information up to time t, if the strategy φ is mean-self-financing. As such it is a mean-variance criterion.*

We introduce the Kunita-Watanabe decomposition: if X is a square-integrable martingale, then every $H \in L_2(P)$ can be written as

$$H = H_0 + \int_0^T \alpha^H(s)dX(s) + N(T) \, , P - \text{ a.s.}$$

where $H_0 = E[H]$ and $N(t)$ is a square-integrable martingale orthogonal to X with $N(0) = 0$.
The risk-minimising strategy is given by

$$\alpha = \alpha^H \, , \beta = V - \alpha \cdot X \tag{28.2.18}$$

with

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$V(t) = H_0 + \int_0^t \alpha^H(s)dX(s) + N(t)$$

Since we are in the martingale case, then

$$V(t) = E[H|\mathcal{F}_t], 0 \leqslant t \leqslant T$$

In that setting, α^H is still as above. The problem is solved by projecting the martingale V associated to H on the martingale X.
In the general incomplete case, we have $P \approx Q$ where P is no-longer a martingale measure. Schweizer [1988] called a strategy locally risk-minimising if, for any $t < T$, the remaining risk in Equation (28.2.17) is minimal under all infinitesimal perturbations of the strategy at time t. This definition is equivalent to the following property of the cost process $C = \{C_t; 0 \leqslant t \leqslant T\}$

***Condition* 2** *The cost process C is a square-integrable martingale orthogonal to M under P*

Definition 28.2.15 *An admissible strategy (α, β) is called optimal if the associated cost process C satisfies the above condition.*

***Remark* 28.2.8** *A unique optimal strategy exists in discrete time which is obtained by a sequential regression procedure running backward from time T to zero.*

It is more difficult to construct such a strategy in continuous time.
Note, an optimal strategy correspond to a Kunita-Watanabe decomposition of the claim where N is orthogonal to the martingale component M of X.

***Proposition* 54** *The existence of an optimal strategy is equivalent to a decomposition*

$$H = H_0 + \int_0^T \alpha^H(s)dX(s) + N(T)$$

with $H_0 \in L_2(\Omega, \mathcal{F}_0, \mathbb{P})$, α^H satisfying some integrability condition, and

$$N = \{N(t); 0 \leqslant t \leqslant T\}$$

is a square-integrable martingale orthogonal to M.
For such a decomposition, the associated optimal strategy is given in Equation (28.2.18).

Thus, the problem of minimising risk is reduced to finding such a representation as in the above Proposition. However, if X is not a martingale, we can not directly use the Kunita-Watanabe projection technique. We can use as a starting point the Kunita-Watanabe decomposition of H with respect to the martingale M

$$H = N(0) + \int_0^T \mu^H(s)dM(s) + (N(T) - N(0))$$

where N is a square-integrable martingale orthogonal to M (not X).
Letting Π be the set of all intrinsic risks, $G(\pi)$ is a measure of intrinsic risk $\pi(\omega)$, mapping from Π to $\mathbb{R}$. The measure G depends on the underlying asset as well as the contingent claim, such that $G^H \in \mathcal{L}^2(\Omega, \mathcal{F}, \mathbb{P})$, leading to the new representation of the claim H as

$$H = V_0 + \int_0^T \alpha(s)dX(s) + G^H \tag{28.2.19}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Given the Kunita-Watanabe decomposition, the measure of intrinsic risk can be expressed as

$$G^H = G_0 + \int_0^T \alpha^H(s)dX(s) + N(T)$$

where $N(t)$ is a square-integrable martingale orthogonal to X. Thus, we have

$$H = V_0^* + \int_0^T \alpha^*(s)dX(s) + N(T)$$

where $V_0^* = V_0 + G_0$ and $\alpha^* = \alpha + \alpha^H$.
Market incompleteness implies the existence of an equivalent measure in the set $\mathcal{Q}$, which is not necessarily a martingale and/or a unique measure.

28.3 Portfolio optimisation

Using the results on stochastic control presented in Appendix (15.4), we are going to detail the optimal portfolio selection problem. We refer the readers to Fleming et al. [1975], Dana et al. [1994], among others. We will follow the notation from Dana et al. [1994].

We consider the continuous model described in Appendix (28.2.2). We let T be the market horizon, and assume $(d+1)$ assets $S = (S_0, S_1, .., S_d)$ traded between 0 and T. We let $S_i(t)$ be the price of the ith asset at time t with price process continuous in time. The asset S_0 is the cash describing the value of \$1 daily compounded at the bank. It is a risk-free asset earning rdt in the time interval $[t, t+dt]$ known at time t. The dynamics of the continuous time model are

$$\begin{aligned} dS_0(t) &= S_0(t)r(t)dt \, , S_0(0) = 1 \\ dS_i(t) &= S_i(t)\big(b_i(t)dt + \sum_{j=1}^d \sigma_{ij}dW_j(t)\big) \end{aligned}$$

where W is a d-dimensional Brownian motion.

28.3.1 Dynamic programming

28.3.1.1 Defining the problem

The strategy consists in choosing between the two different investments, a risk-free one and a risky one. We assume that the strategy (policy) is self-financed and that its consumption in $[0, t]$ is given by $\int_0^t c(s)ds$ where c is a positive process $\mathcal{F}_t$-adapted. The strategy θ finance c, so that the dynamics of the wealth become

$$dX(t) = \sum_{i=0}^d \theta_i(t)dS_i(t) - c(t)dt \, , X(0) = x$$

We let $\pi_i(t) = \theta_i(t)S_i(t)$, $i \geqslant 1$, be the portion of wealth invested in the ith asset, and denote $(\theta S)_t = \pi_t$ the vector with component $(\theta_i(t)S_i(t))_{1 \leqslant i \leqslant d}$ describing the amount to be invested in each stock. Here, the fraction of wealth $\frac{\pi_i(t)}{X(t)}$ for the ith asset corresponds to the stochastic control u_t in Appendix (15.4).
We denote $X_{\pi,c}(t)$ the process of wealth associated to $\{\pi_i; 1 \leqslant i \leqslant d\}$ and the consumption c. Applying Ito's lemma, the dynamics of the agent's wealth are given by

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} dX_{\pi,c}(t) &= \big(X_{\pi,c}(t)r(t) - c(t)\big)dt + \sum_{i=1}^{d} \pi_i\big(b_i(t) - r(t)\big)dt + \sum_{i,j} \pi_i(t)\sigma_{ij}(t)dW_j(t) \\ X_{\pi,c}(0) &= 0 \end{aligned}$$

Some conditions on c and π are imposed to get a unique solution. Eventually, $X_{\pi,c}$ is an Ito process. We denote $\pi(t) = \big(\pi_i(t), ..., \pi_d(t)\big)^\top$ the portfolio of risky assets at time t.
We let U_1 and U_2 be two functions on $\mathbb{R}_+$ and $\mathbb{R}$, respectively, satisfying

U1 U is strictly concave, strictly increasing of class C^1.

U2 $\lim_{x\to\infty} U^{'}(x) = 0$

$(U1)$ shows that function $U^{'}$ is strictly decreasing and admits a continuous inverse, denoted I, defined on $]U^{'}(\infty), U^{'}(0)[$.
For a pair (π, c), we have the performance function

$$J(x;\pi,c) = E[\int_0^T U_1(c(t))dt + U_2(X_{\pi,c}(T))]$$

where E is an expectation under P and $x = X_{\pi,c}(0)$. The agent wants to maximise $J(x;\pi,c)$ under the constraint that his wealth $X_{\pi,c}(t)$ must stay positive.

28.3.1.2 Solving the HJB equation

We assume that r, b, σ are deterministic, and first assume that they are constant. The principle of Dynamic Programming (DP) is as follows: We assume that if we hold the wealth X_α at time α, we know how to optimise wealth between α and T, for all X_α. Thus, we consider the set of all strategies leading to wealth X_α at time α and we select the best one. The principle of DP and all the associated proofs can be found in Appendix (28.3.3).
We introduce an extra parameter to define value at the initial time. Given $\alpha \in]0, T[$, we let $X_{\alpha,x}(t)$ be the wealth of an agent with capital x at time α. To simplify notation we have omitted (π, c). The dynamics are given by

$$\begin{aligned} dX_{\alpha,x}(t) &= \big(X_{\alpha,x}(t)r(t) - c(t)\big)dt + \sum_{i=1}^{d} \pi_i\big(b_i(t) - r(t)\big)dt + \sum_{i,j} \pi_i(t)\sigma_{ij}(t)dW_j(t) \\ X_{\alpha,x}(\alpha) &= 0 \end{aligned}$$

for $\alpha \leqslant t \leqslant T$.
We denote $\mathcal{A}(\alpha, x)$ the set of admissible pairs (π, c). We want to optimise the performance function

$$J(\alpha,x;\pi,c) = E[\int_\alpha^T U_1(c(t))dt + U_2(X_{\alpha,x}(T))]$$

Following Appendix (15.4), we let V be the value function given by

$$V(\alpha,x) = \sup\{J(\alpha,x;\pi,c); (\pi,c) \in \mathcal{A}(\alpha,x)\}$$

The pair (π^*, c^*) is said to be optimal if it is admissible and if

$$J(\alpha,x;\pi^*,c^*) = V(\alpha,x)$$

We can write the principle of DP as follows: For all (α, x), for all $t \geqslant \alpha$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$V(\alpha,x)=\sup_{(\pi,c)\in\mathcal{A}(\alpha,x)}E[\int_{\alpha}^{t}U_1(c(s))ds+V(t,X_{\alpha,x;\pi,c}(t))]$$

Assuming regularity conditions on V, we obtain this function by solving the Hamilton-Jacobi-Bellman equation (see Theorem (15.4.1)) given by

$$\frac{\partial V}{\partial t}+\sup_{(\pi,c)\in\mathbb{R}_+\times\mathbb{R}^d}\Big\{\big[xr(t)-c+\pi^\top(b(t)-r(t)I)\big]\frac{\partial V}{\partial x}+\frac{1}{2}\|\pi^\top\sigma(t)\|^2\frac{\partial^2 V}{\partial x^2}+U_1(c)\Big\}=0\,,\,t\in[0,T[\,,\,x\in\mathbb{R}_{++}$$

where I is the identity matrix, with the boundary conditions

$$\begin{aligned}V(T,x)&=U_2(x)\,,\,x\geqslant 0\\ V(t,0)&=0\,,\,t\in[0,T]\end{aligned}$$

Using the infinitesimal generator $\mathcal{L}$ associated to the diffusion of X, the HJB can be rewritten as

$$\frac{\partial V}{\partial t}(t,x)+\sup_{(\pi,c)}\big(\mathcal{L}V(t,x)+U_1(c)\big)=0 \tag{28.3.20}$$

If we know that there exists an optimal pair (π^*,c^*) and that the value function V satisfies the HJB equation, then we can estimate that pair.
We let the functions I_i be the reciprocal functions of $U_i^{'}$.

Proposition 55 *We assume that there exists an optimal pair (π^*,c^*) and that the value function V satisfies the HJB equation. Then, the pair is*

$$c^*(t)=I_1\big(\frac{\partial V}{\partial x}(t,x)\big)$$

$$\pi^*(t)=-[\sigma\sigma^\top(t)]^{-1}\big(b(t)-r(t)I\big)\frac{\partial V}{\partial x}(t,x)\big[\frac{\partial V^2}{\partial x^2}(t,x)\big]^{-1}$$

The pair (π^*,c^*) is in a feedback form since it is estimated at time t as a function of the optimal wealth at that time. Note,

- in general it is hard to solve explicitly the HJB equation and to get the value function.
- theses results generalise to the case where the utility function is time dependent.
- $\frac{\pi(t)}{\|\pi(t)\|}$ is independent from the utility functions.

Further, we can have solutions to the HJB equation which are not the value function of our problem.

Theorem 28.3.1 *Let v be a function of $C^{1,2}(]0,T[\times\mathbb{R}_+,\mathbb{R}_+)$ satisfying Equation (28.3.20) and the boundary conditions*

$$\begin{aligned}v(T,x)&=U_2(x)\,,\,x\geqslant 0\\ v(t,0)&=0\,,\,t\in[0,T]\end{aligned}$$

Then $V(t,x)\leqslant v(t,x)$, $0\leqslant t<T$, $0\leqslant x<\infty$, where V is the value function.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, in the case where the pair is admissible and has a feedback control, then v is the value function and the pair (π^*, c^*) is optimal.
Note, in the special case where $U_1(x) = x^\alpha$, $0 < \alpha < 1$ and $U_2(x) = 0$, and the parameters r, b, σ are constants, we can completely solve the optimisation problem. In the case of more general utility functions it is much harder to infer the value function. One solution is to use an alternative approach leading to the computation of Cauchy equations rather than HJB equations.

28.3.2 The martingale approach

Using techniques of martingale and concavity, we are going to show the existence of an optimal pair $(c^*, X^*(T))$. Still using martingale techniques we can associate to that pair a portfolio π^* such that the pair (c^*, π^*) is admissible.

28.3.2.1 Admissible strategies

Note, the constraint $X(t) \geqslant 0$ is infinite-dimensional, making it hard to satisfy. We can find an expression such that the trajectories of X no-longer appear, leading to a uni-dimensional constraint. To do so, we need to consider the risk-neutral probability measure. Given

$$\eta(t) = -[\sigma(t)]^{-1}\big(b(t) - r(t)I\big)$$

where I is the identity matrix, and

$$L_t = e^{\int_0^t \eta(s)dW(s) - \frac{1}{2}\int_0^t \|\eta(s)\|^2 ds}$$

and assuming η bounded, we can use Girsanov's theorem. Given $\mathbb{Q}$ the equivalent probability to P defined on $\mathcal{F}_T$ by $dQ = L_T dP$. Then $\tilde{W}(t) = W(t) - \int_0^t \eta(s)ds$ is a $\mathbb{Q} - \mathcal{F}_t$ Brownian motion. The dynamics of the assets, under $\mathbb{Q}$, satisfy

$$dS_i(t) = S_i(t)\big(r(t)dt + \sum_j \sigma_{ij} d\tilde{W}_j(t)\big)$$

and the dynamics of the wealth satisfy

$$dX_{\pi,c}(t) = \big(X_{\pi,c}(t)r(t) - c(t)\big)dt + \sum_{i,j} \pi_i(t)\sigma_{ij}(t)d\tilde{W}_j(t)\ ,\ X_{\pi,c}(0) = x$$

Integrating, we get

$$X_{\pi,c}(t)R(t) = x - \int_0^t c(s)R(s)ds + \int_0^t R(s)\pi^\top(s)\sigma(s)d\tilde{W}(s) \tag{28.3.21}$$

where $R(t)$ is the discounting parameter

$$R(t) = e^{-\int_0^t r(s)ds}$$

Definition 28.3.1 *A pair (π, c) is admissible for an initial wealth x if the process $X_{\pi,c}(t)$ in Equation (28.3.21) is of positive or null values. We denote $\mathcal{A}(x)$ the set of admissible pairs.*

Note, if the pair (π, c) is admissible, the process

$$M_t = x + \int_0^t R(s)\pi^\top(s)\sigma(s)d\tilde{W}(s)$$

is a local Q-martingale equal to $X_{\pi,c}(t)R(t) + \int_0^t c(s)R(s)ds$. It is a positive martingale, hence it is a Q-supermartingale satisfying $E^Q[M_T|\mathcal{F}_0] \leqslant M_0$ so that $E^Q[M_T] \leqslant M_0$.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Proposition 56 *Given an admissible pair (π, c) and the associated final wealth $X_{\pi,c}(T)$. Then*

$$E^Q[X_{\pi,c}(T)R(T) + \int_0^T c(s)R(s)ds] \leqslant x \tag{28.3.22}$$

The converse is true.

Proposition 57 *Given a process c (positive adapted and bounded process) and an $\mathcal{F}_T$ measurable positive variable Z such that*

$$E^Q[ZR(T) + \int_0^T c(s)R(s)ds] = x$$

Then there exists a previsible portfolio π such that the pair (π, c) is admissible, with the associated final wealth $X_{\pi,c}(T)$ being equal to Z. The market is complete.

The constraint in Equation (28.3.22), under P, can be written as

$$E^P[L(T)R(T)X(T) + \int_0^T L(s)R(s)c(s)ds] = x$$

28.3.2.2 Existence of an optimal pair

The problem of maximising the performance $J(x; \pi, c)$ on the set of admissible pairs (π, c) is such that the final wealth $X_{\pi,c}(T)$ be positive and the pair $(c, X_{\pi,c}(T))$ satisfies the constraint

$$E^P[\int_0^T L(t)R(t)c(t)dt + L(T)R(T)X_{\pi,c}(T)] \leqslant x$$

We denote the value function as

$$V(x) = \sup\{J(x; \pi, c); (\pi, c) \in \mathcal{A}(x)\}$$

We want to find a pair $(c^*, X^*(T))$ that maximise $E[\int_0^T U_1(c(t))dt + U_2(X(T))]$ and satisfies the above constraint. We denote the performance function as

$$J(x; \pi, X_{\pi,c}(T)) = E^P[\int_0^T U_1(c(t))dt + U_2(X_{\pi,c}(T))]$$

and we say that the pair $(X(T), c)$ is admissible if the above constraint is satisfied.
We need to present some important properties of the utility functions.

Proposition 58 *If U satisfies $U1$ and $U2$, then*

$$U(I(y)) - yI(y) = \max\big(U(c) - cy, c \geqslant 0\big)$$

As discussed in Appendix (15.4.3), one way to build an optimal pair is to use the multipliers of Lagrange by considering the Lagrangian. For $\lambda \in \mathbb{R}_+$, we denote

$$\mathcal{L}(c, X(T); \lambda) = E[\int_0^T U_1(c(t))dt + U_2(X(T))] + \lambda E[x - (\int_0^T L(t)R(t)c(t)dt + L(T)R(T)X(T))]$$

We want $\lambda^* \in \mathbb{R}_{++}$ such that $(c^*, X^*(T), \lambda^*)$ is a saddle point of $\mathcal{L}$. That is, for all $(c, X(T), \lambda)$, $(c, X(T))$ satisfies the above constraint and $\lambda \in \mathbb{R}_+$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\mathcal{L}(c, X(T); \lambda^*) \leqslant \mathcal{L}(c^*, X^*(T); \lambda^*) \leqslant \mathcal{L}(c^*, X^*(T); \lambda)$$

To satisfy the first inequality we want $(c^*(t), X^*(T))$ such that for all (t, ω), $c^*(t, \omega)$ maximise $U_1(c(t, \omega)) - \lambda^* L(t)R(t)c(t, \omega)$, and for all ω, $X^*(T, \omega)$ maximise $U_2(X(T, \omega)) - \lambda^* L(T)R(T)X(T, \omega)$. We obtain the pair

$$c^*(t) = I_1(\lambda^* \zeta(t)) \text{ and } X^*(T) = I_2(\lambda^* \zeta(T))$$

where

$$\zeta(t) = R(t)L(t)$$

and λ^* should be the constraint is saturated, that is,

$$E[\int_0^T \zeta(t) I_1(\lambda^* \zeta(t))dt + \zeta(T) I_2(\lambda^* \zeta(T))] = x$$

We let the readers refer to Dana et al. [1994] for a check of optimality. To show existence of λ^* we need to make another assumption on the utility functions.

U3

$$\begin{cases} LRI_1(\lambda\zeta) \in L_1(\Omega \times [0, T]; dP \times dt), \forall \lambda > 0 \\ L(T)R(T)I_2(\lambda\zeta(T)) \in L_1(\Omega; dP), \forall \lambda > 0 \end{cases}$$

Lemma 28.3.1 *Under the assumptions $U1, 2, 3$ the function $\mathcal{X}$ defined on $\mathbb{R}_+$ by*

$$\mathcal{X}(y) = E[\int_0^T \zeta(t) I_1(y\zeta(t))dt + \zeta(T) I_2(y\zeta(T))]$$

is strictly decreasing on $[0, \overline{y}]$ where $\overline{y} = \inf\{y | \mathcal{X}(y) = 0\}$, continuous and satisfies

$$\lim_{y \to 0} \mathcal{X}(y) = +\infty \text{ and } \lim_{y \to \infty} \mathcal{X}(y) = 0$$

Thus, the function admits a continuous inverse denoted $\mathcal{Y}$.

We are left with defining $\lambda^* = \mathcal{Y}(x)$. To get the value function we replace c^* and $X^*(T)$ by their values

$$c^*(t) = I_1(\mathcal{Y}(x)\zeta(t)) \text{ and } X^*(T) = I_2(\mathcal{Y}(x)\zeta(T))$$

and

$$V(x) = G(\mathcal{Y}(x))$$

where G is defined on $\mathbb{R}_+$ by

$$G(y) = E[\int_0^T U_1[I_1(y\zeta(t))]dt + U_2[I_2(y\zeta(T))]]$$

We make the assumption:

U4 U_i is of class C^2 and $U_i^{''}$ is increasing, for $i = 1, 2$.

Thus, one can show that G and $\mathcal{X}$ are of class C^1 and $G^{'}(y) = y\mathcal{X}^{'}(y)$. They can both be written as

$$E[\int_0^T \zeta(t) h_1(y\zeta(t))dt + \zeta(T) h_2(y\zeta(T))]$$

for well chosen functions h_i. These functions are associated to solutions of equations of parabolic types.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

28.3.3 Dynamic programming: Some proofs

Dynamic programming (DP) (see Bellman [1957]) belongs to the theory of stochastic control (see Appendix (15.4)) and applies under very general assumptions. It refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner.

28.3.3.1 The principle of dynamic programming

The main idea is as follows:

R1 if we use a strategy on $[\alpha, t]$ and another one on $[t, T]$, we obtain a strategy on $[\alpha, T]$.

R2 if we have a strategy on $[\alpha, T]$, we can decompose it on a strategy on $[\alpha, t]$ and another one on $[t, T]$.

In terms of mathematical optimisation, dynamic programming usually refers to simplifying a decision by breaking it down into a sequence of decision steps over time. This is done by defining a sequence of value functions $V_1, V_2, ..., V_n$ taking y as an argument representing the state of the system at times i from 1 to n. The definition of $V_n(y)$ is the value obtained in state y at the last time n. The values V_i at earlier times $i = n-1, n-2, ..., 2, 1$ can be found by working backwards, using a recursive relationship called the Bellman equation.
In our setting, $[R1]$ leads to

$$E[\int_\alpha^t U_1(c(s))ds + V(t, X_\alpha(t))] \leqslant V(\alpha, x)$$

since $V(t, X_\alpha(t))$ is the optimal strategy on $[t, T]$ and combining a strategy on $[\alpha, t]$ with another one on $[t, T]$ gives a strategy on $[\alpha, T]$.
The equality

$$V(\alpha, x) = \sup_{(\pi,c)\in\mathcal{A}(\alpha,x)} E[\int_\alpha^t U_1(c(s))ds + V(t, X_{\alpha,x;\pi,c}(t))] \tag{28.3.23}$$

comes from the fact that an optimal strategy on $[\alpha, T]$ gives, on $[t, T]$ an optimal strategy.

28.3.3.2 The HJB equation

We can infer the HJB equation from the principle of DP in Equation (28.3.23). To do so we apply Ito's lemma to $V(t, X_{\alpha,x}(t))$ between α and $\alpha + h$, getting

$$V(\alpha+h, X_{\alpha,x}(\alpha+h)) - V(\alpha, x) = \int_\alpha^{\alpha+h} \frac{\partial V}{\partial t}(s, X(s))ds + \int_\alpha^{\alpha+h} \frac{\partial V}{\partial x}(s, X(s))dX_{\alpha,x}(s) + \frac{1}{2}\int_\alpha^{\alpha+h} \frac{\partial^2 V}{\partial x^2}(s, X(s))\|\pi^\top(s)\|^2 ds$$

If we write the principle of DP as

$$V(\alpha, x) \geqslant E[\int_\alpha^{\alpha+h} U_1(c(s))ds + V(\alpha + h, X_{\alpha,x}(\alpha + h))]$$

then, from the above equation, we get

$$\begin{aligned} 0 \geqslant\ & E[\int_\alpha^{\alpha+h} \Big(U_1(c(s)) + \frac{\partial V}{\partial t}(s, X(s)) + \big(X_{\alpha,x}(s)r(s) - c(s) + \pi^\top(s)(b(s) - r(s)I)\big)\frac{\partial V}{\partial x}(s, X(s)) \\ & + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(s, X(s))\|\pi^\top(s)\sigma(s)\|^2\Big)ds + \int_\alpha^{\alpha+h} \pi^\top(s)\sigma(s)\frac{\partial V}{\partial x}(s, X(s))dW(s)] \end{aligned}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

We divide by h and let $h \to 0$, getting

$$0 \geqslant U_1(c(\alpha)) + \frac{\partial V}{\partial t}(\alpha, x) + \big(xr(\alpha) - c(\alpha) + \pi^\top(\alpha)(b(\alpha) - r(\alpha)I)\big)\frac{\partial V}{\partial x}(\alpha, x) + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(\alpha, x)\|\pi^\top(\alpha)\sigma(\alpha)\|^2$$

Since this relation is true for all π, c, we get

$$0 \geqslant \frac{\partial V}{\partial t}(\alpha, x) + \sup_{\pi,c}\Big\{U_1(c) + \big(xr(\alpha) - c + \pi^\top(\alpha)(b(\alpha) - r(\alpha)I)\big)\frac{\partial V}{\partial x}(\alpha, x) + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(\alpha, x)\|\pi^\top(\alpha)\sigma(\alpha)\|^2\Big\}$$

and we obtain the HJB equation. In the case where there exists an optimal pair, the above inequalities become equalities.

28.3.3.3 Sufficient conditions for an optimal solution

Assuming constant parameters and $U_2 = 0$, we state the conditions under which the value function is solution to the HJB. We write $U = U_1$ and let the diffusion operator $\mathcal{L}$ satisfy

$$\mathcal{L}H(t, y) = -\frac{\partial H}{\partial t}(t, y) + ry\frac{\partial H}{\partial y}(t, y) - \frac{1}{2}\|\eta\|^2 y^2 \frac{\partial^2 H}{\partial y^2}(t, y)$$

where $\eta = -\sigma^{-1}(b - rI)$. We assume that there exists functions G and S in $C^{1,3}([0, T] \times \mathbb{R}_{++}, \mathbb{R}_+)$ such that

$$\begin{cases} \mathcal{L}G(t, y) = U(I(y)) \,, t \in [0, T] \,, y \in \mathbb{R}_{++} \\ G(T, y) = 0 \,, y \in \mathbb{R}_{++} \end{cases}$$

and

$$\begin{cases} \mathcal{L}S(t, y) = yI(y) \,, t \in [0, T] \,, y \in \mathbb{R}_{++} \\ S(T, y) = 0 \,, y \in \mathbb{R}_{++} \end{cases}$$

We assume that G, $\frac{\partial G}{\partial y}$, S and $\frac{\partial S}{\partial y}$ satisfy the condition of polynomial growth

$$\max_{0 \leqslant t \leqslant T} H(t, y) \leqslant M\big(1 + y^{-\lambda} + y^{\lambda}\big) \,, y \in \mathbb{R}_{++}$$

where M and λ are strictly positive constants. We also assume that U is of class C^2. One can show the following result:

Theorem 28.3.2 *Under the previous assumptions the value function is of class $C^{1,2}([0, T] \times \mathbb{R}_{++})$ and satisfies the boundary conditions.*

28.4 Introduction to the BSDE

The backward stochastic differential equation (BSDE) were introduced by Bismut [1973] for the linear case, and generalised by Pardoux et al. [1990].

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

28.4.1 Description

The solution of a backward stochastic differential equation (BSDE) is a pair of adapted processes (Y, Z) satisfying

$$-dY_t = f(t, Y_t, Z_t)dt - Z_t^\top dW_t \, , \, Y_T = H \tag{28.4.24}$$

or equivalently

$$Y_t = H + \int_t^T f(s, Y_s, Z_s)ds - \int_t^T Z_s^\top dW_t$$

where f is the generator and H is the terminal condition. The terminal value is an $\mathcal{F}_T$-measurable random variable $H : \Omega \to \mathbb{R}^d$. The generator f maps $\Omega \times \mathbb{R}_+ \times \mathbb{R}^d \times \mathbb{R}^{n\times d}$ onto $\mathbb{R}^d$ and is $\mathcal{P} \otimes \mathcal{B}^d \otimes \mathcal{B}^{n\times d}$-measurable. The solution (Y, Z) is such that $\{Y_t;\ t \in [0, T]\}$ is a continuous $\mathbb{R}^d$-valued adapted process and $\{Z_t;\ t \in [0, T]\}$ is an $\mathbb{R}^{n\times d}$-valued predictable process satisfying $\int_0^T |Z_s|^2 ds < \infty$, P a.s.
If $H \in L_{2,T}(\mathbb{R}^d)$, $f(\cdot, 0, 0) \in \mathbb{H}^2_T(\mathbb{R}^d)$, and f is uniformly Lipschitz, then (f, H) are said to be standard parameters for the BSDE.
The following result is due to Pardoux et al. [1990].

Theorem 28.4.1 *Given standards parameters (f, H), there exists a unique pair $(Y, Z) \in \mathbb{H}^2_T(\mathbb{R}^d) \times \mathbb{H}^2_T(\mathbb{R}^{n\times d})$ solving Equation (28.4.24).*

Proposition 59 *Let $((f_i, H_i);\ i = 1, 2)$ be two standard parameters of the BSDE and $((Y_i, Z_i);\ i = 1, 2)$ be two square-integrable solutions. Let C be a Lipschitz constant for f_1, and put $\delta Y_t = Y_{1,t} - Y_{2,t}$ and $\delta_2 f_t = f_1(t, Y_{2,t}, Z_{2,t}) - f_2(t, Y_{2,t}, Z_{2,t})$. For any (λ, μ, β) such that $\mu > 0$, $\lambda^2 > C$ and $\beta \geqslant C(2 + \lambda^2) + \mu^2$, it follows that*

$$\|\delta Y\|^2_\beta \leqslant T\big[e^{\beta T} E[|\delta Y_T|^2] + \frac{1}{\mu^2}\|\delta_2 f\|^2_\beta\big]$$

and

$$\|\delta Z\|^2_\beta \leqslant \frac{\lambda^2}{\lambda^2 - C}\big[e^{\beta T} E[|\delta Y_T|^2] + \frac{1}{\mu^2}\|\delta_2 f\|^2_\beta\big]$$

Note, in the control of the norm of δT, we can replace T by $\inf\big(T, [\beta - (C(2+\lambda^2) + \mu^2)]^{-1}\big)$. Further, one can prove that

$$E[\sup_{t\leqslant T} |\delta Y_t|^2] \leqslant KE[|\delta Y_T|^2 + \int_0^T |\delta_2 f_t|^2 dt]$$

where K is a positive constant depending only on T. We can then derive that the Picard iterative sequence converges almost surely to the solution of the BSDE.

Corollary 8 *Let β be such that $2(1+T)C < \beta$. Let (Y_k, Z_k) be the sequence defined recursively by $(Y_0 = 0, Z_0 = 0)$ and*

$$-dY_{k+1,t} = f(t, Y_{k,t}, Z_{k,t})dt - (Z_{k+1,t})^\top d\widehat{W}_t \, , \, Y_{k+1,T} = H$$

Then the sequence (Y_k, Z_k) converges to (Y, Z), $dP \otimes dt$ a.s. as k goes to ∞.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

28.4.1.1 The linear BSDE

We now discus linear BSDE (LBSDE).

Proposition 60 *Let (β, γ) be a bounded $(\mathbb{R}, \mathbb{R}^n)$-valued predictable process, φ an element of $\mathbb{H}^2_T(\mathbb{R})$, and H an element of $L_{2,T}(\mathbb{R})$. Then the LBSDE*

$$-dY_t = \big(\varphi_t + Y_t\beta_t + Z_t^\top \gamma_t\big)dt - Z_t^\top d\widehat{W}_t \, , \, Y_T = H$$

has a unique solution (Y, Z) in $\mathbb{H}^2_{T,\beta}(\mathbb{R}) \times \mathbb{H}^2_{T,\beta}(\mathbb{R}^n)$ and Y_t is given by the closed formula

$$Y_t = E\big[H\Gamma_{t,T} + \int_t^T \Gamma_{t,s}\varphi_s ds|\mathcal{F}_t\big] \, , \, \mathbb{P} \text{ a.s.}$$

where $\Gamma_{t,s}$ is the adjoint process defined for $s \geqslant t$ by the forward LSDE

$$\frac{d\Gamma_{s,t}}{\Gamma_{s,t}} = \beta_s ds + \gamma_s^\top d\widehat{W}_s \, , \, \Gamma_{t,t} = 1$$

In particular, if H and φ are non-negative, the process Y is non-negative. If, in addition, $Y_0 = 0$, then for any t, $Y_t = 0$ a.s., and $\varphi_t = 0$ $dP \otimes dt$ a.s.

We can use the above proposition to derive a comparison theorem.

Theorem 28.4.2 *Comparison theorem*
Let (f_i, H_i), $i = 1, 2$, be two standard parameters of BSDEs, and let (Y_i, Z_i), $i = 1, 2$, be the associated square-integrable solutions. We suppose that $H_1 \geqslant H_2$, $\mathbb{P}$ a.s. and

$$\delta_2 f_t = f_1(t, Y_{2,t}, Z_{2,t}) - f_2(t, Y_{2,t}, Z_{2,t}) \geqslant 0 \, , \, d\mathbb{P} \otimes dt \text{ a.s}$$

Then, we have almost surely for any time t, $Y_{1,t} \geqslant Y_{2,t}$.

We can then deduce a sufficient condition for the non-negativity of the BSDE solution.

Corollary 9 *If $H \geqslant 0$ a.s. and $f(t, 0, 0) \geqslant 0$, $d\mathbb{P} \otimes dt$ a.s., then $Y \geqslant 0$ $\mathbb{P}$ a.s. Further, if $Y_t = 0$ on a set $A \in \mathcal{F}_t$, then $Y_s = 0$, $f(s, 0, 0) = 0$ on $[t, T] \times A$, $d\mathbb{P} \otimes ds$ a.s., and $H = 0$ almost surely on A.*

The authors showed that a superhedging strategy was a supersolution of a one-dimensional LBSDE.

Definition 28.4.1 *Suppose that $d = 1$. A supersolution of a BSDE associated with standard parameters (f, H) is a vector process (Y, Z, C) satisfying*

$$-dY_t = f(t, Y_t, Z_t)dt - Z_t^\top d\widehat{W}_t + dC_t \, , \, Y_T = H$$

or equivalently

$$Y_t = H + \int_t^T f(s, Y_s, Z_s)ds - \int_t^T Z_s^\top d\widehat{W}_s + \int_t^T dC_s$$

where H is an $\mathbb{R}$-valued $\mathcal{F}_T$-measurable random variable, $(Y_t;\, t \in [0, T])$ is a right-continuous, left-limited adapted real process, Z is a predictable process taking values in $\mathbb{R}^n$, $(C_t;\, t \in [0, T])$ is an increasing, adapted, right-continuous process with $C_0 = 0$, Y is bounded below.

Note, if (f, H) are standard parameter, the continuous supersolutions dominate the classical square-integrable solution of the BSDE.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

28.4.1.2 The flow property

The properties of continuity and differentiability of the solutions of BSDEs depends on parameters and the flow properties of a BSDE. We denote the dependence of a solution of a BSDE with respect to terminal condition by $(Y(T,H), Z(T,H))$. The flow property is as follows:

***Proposition* 61** *Let (Y,Z) be the solution of a BSDE with standard parameters (T, f, H).*

- *For any stopping time $S \leqslant T$,*

$$Y_t(T,H) = Y_t(S, Y_S(H))\,,\; Z_t(T,H) = Z_t(S, Y_S(H))\,,\; t \in [0,S]\,,\; d\mathbb{P} \otimes dt \text{ a.s.}$$

- *Suppose that the sequence of stopping times S_n converges a.s. to S and that the sequence of the terminal variables $H_n \in \mathcal{F}_{S_n}$ converges in $L_{2,T}(\mathbb{R}^d)$ to $H \in \mathcal{F}_s$. Then the pair of processes $(Y(S_n, H_n), Z(S_n, H_n))$ converges in $\mathbb{H}^2_{T,\beta}(\mathbb{R}^d) \times \mathbb{H}^2_{T,\beta}(\mathbb{R}^{n\times d})$ to $(Y(S,H), Z(S,H))$.*

28.4.2 Control problems

We can also consider the solution of a BSDE with respect to standard parameters that are infima of standard parameters. That is,

$$Y_t(\inf f_\alpha, \inf H_\alpha) = \operatorname{ess\,inf} Y_t(f_\alpha, H_\alpha)$$

It can be applied to some classical control problems. Under some mild conditions, the value function can be characterised as the solution of a concave BSDE. Conversely, the solution (Y,Z) of a BSDE (with $d = 1$) with concave generator can be considered as the value function of a control problem.
From the comparison theorem (see Theorem (28.4.2)), the solution of a BSDE with standard parameters (f, H) is less than the infimum of solutions associated with (f_α, H_α). The following proposition define when the equality holds.

***Proposition* 62** *Let (f,H) and (f_α, H_α) be a family of standard parameters, and let (Y,Z) and (Y_α, Z_α) be the solution of the associated BSDEs. Suppose that there exists a parameter $\overline{\alpha}$ such that*

$$f(t, Y_t, Z_t) = \operatorname{ess\,inf}_\alpha f_\alpha(t, Y_t, Z_t) = f_{\overline{\alpha}}(t, Y_t, Z_t)\,,\; d\mathbb{P} \otimes dt \text{ a.s.}$$

and

$$H = \operatorname{ess\,inf}_\alpha H_\alpha = H_{\overline{\alpha}}\,,\; d\mathbb{P} \otimes dt \text{ a.s.}$$

Then the process Y and Y_α satisfy

$$Y_t = \operatorname{ess\,inf}_\alpha Y_{\alpha,t} = Y_{\overline{\alpha},t}\,,\; \forall t \in [0,T]\,,\; \mathbb{P} \text{ a.s.}$$

Some stochastic control problems (see Davis [1973], El Karoui [1981]) can be formulated as follows: The laws of the controlled process belong to a family of equivalent measures with densities satisfying

$$\frac{dR_{u,t}}{R_{u,t}} = d(t,u_t)dt + n(t,u_t)^\top d\widehat{W}_t\,,\; R_{u,0} = 1$$

where $d(t,u_t)$ and $n(t,u_t)$ are predictable processes uniformly bounded by δ_t and ν_t, respectively. A feasible control $(u_t;\; t \in [0,T])$ is a predictable process valued in a space U. The set of feasible controls is denoted by $\mathcal{U}$. We want to minimise over all feasible control processes u the objective function

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$J(u) = E[\int_0^T R_{u,t}k(t,u_t)dt + R_{u,T}K(u_T)]$$

where $K(\cdot, u_T)$ is the terminal condition and $k(\cdot, t, u_t)$ is the running cost associated with the control process u. The controller acts on the law of processes by change of equivalent probability measures with Radon-Nikodym derivatives given by

$$dL_{u,s} = L_{u,s}n(s,u_s)^\top d\widehat{W}_s$$

and by a controlled discount factor with bounded rate $d(s,u_s)$. That is,

$$dD_{u,s} = D_{u,s}d(s,u_s)ds \text{ with } R_{u,t} = D_{u,t}L_{u,t}$$

We let $\mathbb{Q}_u$ be the probability measure with density $L_{u,T}$ on $\mathcal{F}_T$. Then, we can write the objective function as

$$J(u) = E^{Q_u}[\int_0^T D_{u,t}k(t,u_t)dt + D_{u,T}K(u_T)]$$

28.4.2.1 Value function expressed as BSDE

By Proposition (60), $J(u) = Y_{u,0}$, where (Y_u, Z_u) is the solution of the linear BSDE associated with standard parameters (f_u, H_u), where

$$f_u(t,y,z) = k(t,u_t) + d(t,u_t)\cdot y + n(t,u_t)^\top z \text{ , } H_u = K(u_T)$$

The process R_u corresponds to the adjoint process associated with (Y_u, Z_u) and

$$Y_{u,t} = E[\int_0^T R_{u,t,s}k(s,u_s)ds + R_{u,t,T}H_u|\mathcal{F}_t]$$

We can use this result to obtain the verification theorem as follows:

***Proposition* 63** *Verification theorem*
The parameters (f,H) defined by

$$f(t,y,z) = \operatorname{ess\,inf}\{f_u(t,y,z)|\ u\in\mathcal{U}\} \text{ , } H = \operatorname{ess\,inf}\{H_u|\ u\in\mathcal{U}\}$$

are standard parameters. Let (Y,Z) be the solution of the BSDE associated with terminal condition H. Then Y is the value function Y^* of the control problem. That is, for each $t\in[0,T]$,

$$Y_t = Y^* \operatorname{ess\,inf}\{Y_{u,t}|\ u\in\mathcal{U}\}$$

The principle of dynamic programming in stochastic control corresponds to the flow property in BSDEs. Using the same notation, $S\leqslant T$ is a stopping time, H_s is an $\mathcal{F}_s$-measurable variable, and $Y_{u,t}(S,H_s)$ is the solution of a BSDE with standard parameters $(T, f_u(t,y,z))I(t\leqslant S), H_s)$.

***Proposition* 64** *The value function Y_t satisfies the dynamic programming principle: for any time t and any stopping time S with $t\leqslant S\leqslant T$,*

$$Y_t(T,H) = \operatorname{ess}\inf_{u\in\mathcal{U}} Y_{u,t}(S, Y_S(T,H)) \text{ , } \mathbb{P} \text{ a.s.}$$

which can also be written

$$Y_t(T,H) = \operatorname{ess}\inf_{u\in\mathcal{U}} E[\int_t^S R_{u,t,s}k(s,u_s)ds + R_{u,t,S}Y_S(T,H)|\mathcal{F}_t] \text{ , } \mathbb{P} \text{ a.s.}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

The optimisation problem is to find a 0-optimal control u_0 reaching the minimum for the problem $\inf\{Y_{u,0}|\ u \in \mathcal{U}\}$. That is, $Y_0^* = Y_{u_0,0}$.
The following Corollary is a criterion for finding 0-optimal controls.

***Corollary* 10** *Optimality criterion*
A control $(u_{0,s};\ 0 \leqslant s \leqslant T)$ *is* 0*-optimal if and only if*

$$f(s, Y_s, Z_s) = f_{u_0}(s, Y_s, Z_s)\ ,\ d\mathbb{P} \otimes ds \text{ a.s.}$$

and

$$H = H_{u_0}\ ,\ \mathbb{P} \text{ a.s.}$$

In this case, u_0 *is also optimal for the problem starting at time* t*, that is,* $Y_t^* = Y_{u_0,t}$.

28.4.2.2 BSDE expressed as value function

We are now going to see that a concave BSDE is always associated with a control problem. We let $f(t, y, z)$ be a standard generator of a BSDE, concave with respect to y, z, and we let $F(t, \beta, \gamma)$ be the polar process associated with f:

$$F(\omega, t, \beta, \gamma) = \sup_{(y,z)\in\mathbb{R}\times\mathbb{R}^n} \{f(\omega, t, y, z) - \beta y - \gamma^\top z\}$$

The effective domain of F is given by

$$\mathcal{D}_F = \{(\omega, t, \beta, \gamma) \in \Omega \times [0, T] \times \mathbb{R} \times \mathbb{R}^n |\ F(\omega, t, \beta, \gamma) < \infty\}$$

We want to associate with F a wide enough family of linear standard generators $f_{\beta,\gamma}$ such that Proposition (62) holds. We let

$$f_{\beta,\gamma}(t, y, z) = F(t, \beta_t, \gamma_t) + \beta_t y - \gamma_t^\top z$$

where (β, γ) are predictable processes, called control parameters. By the conjugacy relation, f is also the infimum of $f_{\beta,\gamma}$. To ensure that $f_{\beta,\gamma}$ is a standard generator, we need to assume that (β, γ) belongs to $\mathcal{A}$ defined by

$$\mathcal{A} = \{(\beta, \gamma) \in \mathcal{P}, K - \text{ valued } |\ E[\int_0^T F(t, \beta_t, \gamma_t)^2 dt < \infty]\}$$

where $\mathcal{A}$ is called the set of admissible control parameters. To apply Proposition (62) we need the following lemma.

Lemma 28.4.1 *There exists an optimal control* $(\overline{\beta}, \overline{\gamma}) \in \mathcal{A}$ *such that*

$$f(t, Y_t, Z_t) = f_{\overline{\beta},\overline{\gamma}}(t, Y_t, Z_t)\ ,\ d\mathbb{P} \otimes dt \text{ a.s.}$$

For each control process $(\beta, \gamma) \in \mathcal{A}$ we introduce the dual controlled objective processes $(Y_{\beta,\gamma}, Z_{\beta,\gamma})$ as the unique solution of the LBSDE with data $(f_{\beta,\gamma}, H)$. Then, Proposition (62) gives the following result:

***Proposition* 65** *Let* f *be a concave standard generator and* $f_{\beta,\gamma}$ *the associated linear standard generator satisfying*

$$f = \operatorname{ess\,inf}\{f_{\beta,\gamma}|\ (\beta, \gamma) \in \mathcal{A}\}\ ,\ d\mathbb{P} \otimes dt \text{ a.s.}$$

Then $\mathbb{P}$ *a.s. for any time* t*,*

$$Y_t = \operatorname{ess\,inf}\{Y_{t,\beta,\gamma}|\ (\beta, \gamma) \in \mathcal{A}\}$$

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

This result is associated to a control problem with control set $\mathcal{A}$. The solution $Y_{t,\beta,\gamma}$ from the LBSDE can be expressed in terms of the adjoint process $(\Gamma_{\beta,\gamma}(t,s);\ t \leqslant s \leqslant T)$ which is the unique solution of the linear SDE

$$\frac{d\Gamma_s}{\Gamma_s} = \beta_s ds + \gamma_s d\widehat{W}_s \ , \Gamma_t = 1$$

in the following way:

$$Y_{t,\beta,\gamma} = E[\int_t^T \Gamma_{\beta,\gamma}(t,s)F(s,\beta_s,\gamma_s)ds + \Gamma_{\beta,\gamma}(t,T)H|\mathcal{F}_y]$$

where $Y_{\beta,\gamma}$ is called the controlled objective function of a control problem with running cost $F(t,\beta,\gamma)$ and terminal cost H.

28.4.2.3 Variational formulation of recursive utility

Assuming concavity of the generator f with respect to (c,y,z), the recursive utility can be written as the value function of a control problem. Given the terminal reward H taking value in $L_{2,T}(\mathbb{R})$, c a consumption process in $\mathbb{H}_T^2$, we let $Y_{c,H}$ be the recursive utility: the solution of the BSDE associated with generator $f(t,c_t(\cdot),\cdot,\cdot)$ and terminal value H. For a consumption rate c_t, we let $F(t,c_t,\cdot,\cdot)$ be the polar function of $f(t,c_t,\cdot,\cdot)$, that is,

$$F(t,c_t,\beta,\gamma) = \sup_{(y,z)\in\mathbb{R}\times\mathbb{R}^n} \{f(t,c_t,y,z) - \beta\cdot y - \gamma\cdot z\}$$

We let $\mathcal{A}(c)$ be the set of admissible processes (β,γ) such that $E[\int_0^T F^2(t,c_t,\beta,\gamma)dt < \infty]$. Then the recursive utility can be written as

$$Y_{t,c,H} = \text{ess}\inf_{(\beta,\gamma)\in\mathcal{A}} E[\int_t^T \Gamma_{\beta,\gamma}(t,s)F(s,c_s,\beta_s,\gamma_s)ds + \Gamma_{\beta,\gamma}(t,T)H|\mathcal{F}_t]$$

Thus, we see that the recursive utility $Y_{c,H}$ can be defined through a felicity function F expressed in terms of the current consumption c, the current rate $-\beta$, and the risk premium $-\gamma$. Further, the adjoint processes $\Gamma_{\beta,\gamma}$ can be interpreted as a deflator. Also, the process $Y_{t,\beta,\gamma}$ can be interpreted as an ex post utility when the deflator is given by $\Gamma_{\beta,\gamma}(t)$. Thus, the utility is equal to the minimum of ex post utilities over all price deflators.

28.4.3 Superstrategy

We consider the continuous-time asset pricing framework with $n+1$ assets described in Appendix (28.2.2). The risk-free asset S_0 has dynamics

$$dS_0(t) = r_t S_0(t)dt$$

where r_t is the bounded short rate. The dynamics of the ith risky asset satisfies

$$\frac{dS_i(t)}{S_i(t)} = b_i(t)dt + \sum_{j=1}^n \sigma_{i,j}(t)d\widehat{W}_j(t)$$

where $\widehat{W} = (\widehat{W}_1,...,\widehat{W}_n)^\top$ is a standard Brownian motion on $\mathbb{R}^n$ under the $\mathbb{P}$-measure. Here, the volatility matrix is of size $n\times n$. The market is assumed complete.
We let the contingent claim H, settled at time T, be an $\mathcal{F}_T$-measurable random variable. We consider a small investor whose actions cannot affect market prices. In the Merton model (see Merton [1971]), he can decide at time $t\in[0,T]$ the amount $\pi_i(t)$ of the wealth V_t to invest in the ith stock. He can also choose his consumption c_t. We let $C_t = \int_0^t c_s ds$ be the cumulative amount of consumption between 0 and t. It is the adapted increasing consumption process. It can

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

also be interpreted as the necessary liquidity to satisfy some constraints. At time $t = T$, the agent wants to be certain that he will cover his obligation, that is, $V_T \geqslant H$ will hold almost surely. He needs to find the smallest value of $x > 0$ for which such hedging is possible. The latter is the price of the contract at time $t = 0$.
In the Merton model, we get the following definition:

Definition 28.4.2 *Self-financing superstrategy*
A self-financing superstrategy is a vector process (V, π, C), where V is the market value, C the cumulative consumption process, and $\pi = (\pi_1, ..., \pi_n)^\top$ is the portfolio process, with dynamics

$$dV_t = r_t V_t dt - dC_t + \pi_t^\top \sigma_t \big(d\widehat{W}_t + \lambda_t dt\big) \ , \ \int_0^T |\sigma_t^\top \pi_t|^2 dt < \infty \ \mathbb{P} \ a.s.$$

where C is an increasing, right-continuous, adapted process with $C_0 = 0$.

The superstrategy is feasible if the constraint of non-negative wealth holds

$$V_t \geqslant 0 \ , t \in [0, T] \ , \ \text{a.s.}$$

Then, superhedging strategy is defined as follows:

Definition 28.4.3 *Let $H \geqslant 0$ be a positive contingent claim.*

1. *A superhedging strategy against H is a feasible self-financing strategy (V, π, C) such that $V_T = H$. $\mathcal{H}^{'}$ is the class of hedging strategies against H. If $\mathcal{H}^{'}$ is non-empty, H is called superhedgeable.*
2. *The fair price $X_0^{'}$ at time 0 of the superhedgeable claim H is the smallest initial endowment needed to hedge H. That is,*

$$X_0^{'} = \inf\{x \geqslant 0; \ \exists (V, \pi, C) \in \mathcal{H}^{'} \text{ such that } V_0 = x\}$$

In a complete market, the fair price is the market value of a hedging strategy in $\mathcal{H}$. This is stated in the following theorem:

Theorem 28.4.3 *Given some assumptions, let H be a positive square-integrable contingent claim. There exists a hedging strategy (X, π) against H such that*

$$dX_t = r_t X_t dt + \pi_t^\top \sigma_t \lambda_t dt + \pi_t^\top \sigma_t \lambda_t d\widehat{W}_t \ , \ X_T = H \tag{28.4.25}$$

and such that the market value X is the fair price and the upper price of the claim.
We let $(D_{t,s}; \ s \geqslant t)$ be the deflator started at time t and given by

$$dD_{t,s} = -D_{t,s}\big(r_s ds + \lambda_s^\top d\widehat{W}_s\big) \ , \ D_{t,t} = 1$$

Then

$$X_t = E[D_{t,T} H | \mathcal{F}_t] \ a.s.$$

Duffie et al. [1992b] [1992c] defined the deflator as the process

$$D_t = e^{-\left(\int_0^t r_s ds + \int_0^t \lambda_s^\top d\widehat{W}_s + \frac{1}{2}\int_0^t |\lambda_s|^2 ds\right)}$$

which is the solution of the above SDE started at time 0.
The above equation corresponds to the property that the fair price of H can be computed as the expectation of the discounted value of the claim under the risk-neutral probability measure (see Equation (22.1.3)). That is,

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$X_t = E^Q[e^{-\int_t^T r_s ds} H|\mathcal{F}_t]$$

where $\mathbb{Q}$ is the risk-neutral probability.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

Bibliography

[1989] Aarst E.H.L., Korst J., Simulated annealing and Boltzmann machines. John Wiley.

[2019] Abati D., Porrello A., Calderara S., Cucchiara R., Latent space autoregression for novelty detection. In *CVPR*, pp 481–490.

[2012] Abernethy J., Chen Y., Vaughan J.W., Efficient market making via convex optimization, and a connection to online learning. Working Paper, University of California, Berkeley.

[1993] Abry P., Goncalves P., Flandrin P., Wavelet-based spectral analysis of $1/f$ processes. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, **3**, pp 237–240.

[1995] Abry P., Goncalves P., Flandrin P., Wavelets, spectrum estimation and $1/f$ processes. in A. Antoniadis and G. Oppenheim, eds, *Lecture Notes in Statistics: Wavelets and Statistics*, Springer-Verlag, pp 15–30.

[1996] Abry P., Sellan F., The wavelet-based synthesis for fractional Brownian motion. proposed by F. Sellan and Y. Meyer, Remarks and Fast Implementation, *Applied and Computational Harmonic Analysis*, **3**, (4), pp 377–383.

[1998] Abry P., Veitch D., Wavelet analysis of long-range-dependent traffic. *IEEE Transaction on Information Theory*, **44**, (1), pp 2–15.

[1999] Abry P., Sellan F., A wavelet-based joint estimator of the parameters of long-range dependence. *IEEE Transaction on Information Theory*, **45**, (3), pp 878–897.

[2000] Abry P., Flandrin P., Taqqu M.S., Veitch D., Wavelets for the analysis, estimation, and synthesis of scaling data. in *Self-similar Network Traffic and Performance Evaluation*, ed. K. Park, W. Willinger, Wiley, pp 39–87.

[2002] Abry P., Baraniuk R., Flandrin P., Riedi R., Veitch D., Multiscale nature of network traffic. *IEEE Signal Processing Magazine*, **19**, (3), pp 28–46.

[2001] Abu-Mostafa Y.S., Atiya A.F., Magdon-Ismail M., White H., Neural networks in financial engineering. *IEEE Transactions on Neural Networks*, **12**, (4), pp 653–656.

[2012] Abu-Mostafa Y.S., Magdon-Ismail M., Lin H-T., Learning from data. AMLBook.

[1994] Accardi L., De Finetti theorem. In M. Hazewinkel, ed,, Encyclopedia of Mathematics, Springer Science.

[1998] Adya M., Collopy F., How effective are neural networks at forecasting and prediction? A review and evaluation. *Journal of Forecasting*, **17**, pp 481–495.

[2004] Agarwal V., Naik N.Y., Risk and portfolio decisions involving hedge funds. *The Review of Financial Studies*, **17**, (1), pp 63–98.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2017] Aghdam H.H., Heravi E.J., Guide to convolutional neural networks: A practical application to traffic-sign detection and classification. Springer.

[1993] Agrawal R., Imielinski T., Swami A., Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pp 207.

[2006] Aharon M., Elad M., Bruckstein A., K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing*, IEEE Transactions on 54, (11), pp 4311–4322.

[2010] Ahmed N.K., Atiya A.F., El Gayar N., El-Shishiny H., An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, **29**, (5), pp 594–621.

[2018] Akcay S., Atapour-Abarghouei A., Breckon T.P., GANomaly: Semi-supervised anomaly detection via adversarial training. In *ACCV*, springer, pp 622–637.

[1999] Albert R., Jeong H., Barabasi A-L., Diameter of the world-wide web. *Nature*, **401**, pp 130–31.

[2000] Albert R., Barabasi A-L., Topology of evolving networks: Local events and universality. *Physical Review Letters*, American Physical Society (APS), **85**, (24), pp 5234–5237.

[2002] Albert R., Barabasi A-L., Statistical mechanics of complex networks. *Review of Modern Physics*, **74**, (1), pp 47–97.

[1981] Albus J.S., Brains, behaviour and robotics. BYTE Books, McGraw-Hill.

[2016] Alenazi S.R., Kamsuriah A., Record duplication detection in database: A review. *International Journal on Advanced Science Engineering and Information Technology*, **6**, (6), 838.

[2008] Alexander C., Market risk analysis: Practical financial econometrics. John Wiley & Sons, Ltd., Chichester.

[1995] Ali K., Learning probablistic relational concept descriptions. Thesis, Computer Science, University of California, Irvine.

[2001] Allison P.D., Missing data. SAGE Publications.

[2008] Alonso J-M., Chen Y., Receptive field. *Scholarpedia*, **4**, (1), 5393.

[2004] Alos E., A generalization of Hull and White formula and applications to option pricing approximation. Working Paper, Universitat Pompeu Fabra.

[2007] Alos E., Ewald CO., Malliavin differentiability of the Heston volatility and applications to option pricing. Working Paper, Universitat Pompeu Fabra, University of St.Andrews.

[2010] Alpaydin E., Introduction to machine learning. MIT Press

[1997] Amit Y., Geman D., Shape quantization and recognition with randomized trees. *Neural Computation*, **9**, (7), pp 1545–1588.

[1996] Anders U., Korn O., Schmitt C., Improving the pricing of options: a neural network approach. ZEW Discussion Papers, pp 96–104.

[1986] Anderson C.W., Learning and problem solving with multilayer connectionist systems. Doctoral dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst.

[1987] Anderson C.W., Strategy learning with multilayer connectionist representations. In *Proceedings of the fourth international workshop on Machine Learning*, pp 103–114, Ann Arbor, Michigan, Morgan Kaufmann.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1994] Andre D., Evolution of mapmaking ability: Strategies for the evolution of learning, planning, and memory using genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1. IEEE Press, June.

[1996] Andre D., Bennett F.H., Koza J.R., Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp 3–11, Stanford University, CA, USA. MIT Press.

[2016] Andreas J., A neural network is a monference, not a model. http://blog.jacobandreas.net/monference.html.

[2010] Androutsopoulos I., Malakasiotis P., A survey of paraphrasing and textual entailment methods. *Journal of Artificial Intelligence Research*, **38**, pp 135–187.

[1993] Angeline P.J., Pollack J.B., Competitive environments evolve better solutions for complex tasks. In *Proceedings of the 5th International Conference on Genetic Algorithms*, ICGA-93, pp 264–270. Morgan Kaufmann.

[1994] Angeline P.J., Saunders G.M., Pollack J.B., An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, **5**, (1), pp 54–65.

[2008] Angles R., Gutierrez C., Survey of graph database models. *ACM Computing Surveys*, **40**, (1), pp 1–39.

[2015] Antonik P., Duport F., Smerieri A., Haelterman P., Massar S., FPGA implementation of reservoir computing with online learning. Université Libre de Bruxelles.

[2016] Antonik P., Hermans M., Duport F., Haelterman M., Massar S., Towards pattern generation and chaotic series prediction with photonic reservoir computers. Laboratoire d'Information Quantique, Universite Libre de Bruxelles, Belgium.

[1974] Appel G., Double your money every three years. Brightwaters, NY: Windsor Books.

[1999] Appel G., Technical analysis power tools for active investors. Financial Times Prentice Hall.

[2008] Appel G., Appel M., A quick tutorial in MACD: Basic concepts. Working Paper.

[2006] Applegate D.L., Bixby R.E., Chvatal V., Cook W.J., Concorde tsp solver. Working Paper.

[2011] Applegate D.L., Bixby R.E., Chvatal V., Cook W.J., The traveling salesman problem: A computational study. Princeton university press, 2011.

[2017] Arjovsky M., Chintala S., Bottou L., Wassertein generative adversarial networks. In *ICML*, pp 214–223.

[2001] Armstrong J.S., Principles of forecasting: A handbook for researchers and practitioners. ed.: Norwell, MA: Kluwer Academic Publishers.

[1991] Arneodo A., Bacry E., Muzy J-F., Wavelets and multifractal formalism for singular signals: Application to turbulence data. *Phys. Rev. Lett.*, **67**, pp 3515–3518.

[1995] Arneodo A., Bacry E., Graves P.V., Muzy J-F., Characterizing long-range correlations in DNA sequences from wavelet analysis. *Phys. Rev. Lett.*, **74**, 3293.

[1998] Arneodo A., Muzy J-F., Dornette D., Discrete causal cascade in the stock market. *Eur. Phys. J.*, **2**, pp 277–282.

[2017] Arras L., Montavon G., Muller K-R., Samek W., Explaining recurrent neural network predictions in sentiment analysis. In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pp. 159–168.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1989] Ash T., Dynamic node creation in backpropagation networks. *Connection Science*, **1**, (4), pp 365–375.

[2007] Aslam J.A., Popa R.A., Rivest R.L., On estimating the size and confidence of a statistical audit. In *Proceedings of the Electronic Voting Technology Workshop (EVT '07)*, Boston, MA, August 6, 2007.

[2000] Atiya A.F., Parlos A.G., New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on neural networks*, **11**, (3), pp 697–709.

[1999] Auda G., Kamel M., Modular neural networks: A survey. *International Journal Neural Systems*, **9**, (2), pp 129–151.

[1996] Aytug H., Koehler G.J., New stopping criterion for genetic algorithms. Working Paper, University City Blvd and University of Florida.

[2019] Babbar K., McGhee W.A., A deep learning approach to exotic option pricing under LSVol. Working Paper.

[2001] Bacry E., Delour J., Muzy J-F., Multifractal random walk. *Physical Review E*, **64**, 026103–026106.

[2015] Bahdanau D., Cho K., Bengio Y., Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015. Also, arXiv:1409.0473, 2014.

[2006] Bai R., Wang X., An effective hybrid classifier based on rough set and neural network. *International Conference IEEE*, 0-7695-2749-3/06.

[2018] Bai S., Kolter J.Z., Koltun V., An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv:1803.01271.

[1996] Baillie R.T., Long memory processes and fractional integration in econometrics. *Journal of Econometrics*, **73**, pp 5–59.

[1996] Baillie R.T., Bollerslev T., Mikkelsen H.O., Fractionally integrated generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, **74**, pp 3–30.

[1975] Baker J.K., The dragon system: An overview. *IEEE Trans. Acoust. Speech Signal Processing*, **23**, (1), pp 24–29.

[1985] Baker J.E., Adaptive selection methods for genetic algorithms. *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, pp 101–111.

[2017] Baker B., Gupta O., Naik N., Raskar R., Designing neural network architectures using reinforcement learning. In *Proceedings of the 5th International Conference on Learning Representations*, (ICLR 17), Toulon, France.

[2001] Balabanian N., Carlson B., Digital logic design principles. John Wiley.

[2007] Balamurugan R., Subramanian S., Self-adaptive differential evolution based power economic dispatch of generators with valve-point effects and multiple fuel options. *International Journal of Computer Science and Engineering*, Winter.

[2016] Ban G-Y., El Karoui N., Lim A.E.B., Machine learning and portfolio optimization. *Management Science Articles in Advance*.

[2009] Bandurski K., Kwedlo W., Training neural networks with hybrid differential evolution algorithm. Zeszyty Naukowe Politechniki Bialostockiej.

[2010] Bang-Jensen J., Gutin G,Z., Digraphs: Theory, algorithms and applications. Sringer Monographs in Mathematics, 2 ed., Sringer-Verlag.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2004] Bansal N., Blum A., Chawla S., Correlation clustering. *Machine Learning*, **56**, (1-3), pp 89–113.

[1991] Barabasi A-L., Vicsek T., Multifractality of self-affine fractals. *Physical Review A*, **44**, (4), pp 2730–2733.

[1999] Barabasi A-L., Emergence of scaling in random networks. *Science*, **286**, pp 509–515.

[1999] Barabasi A-L., Albert R., Jeong H., Mean-field theory for scale-free random networks. *Physica A*, **272**, pp 173–187.

[2001] Barabasi A-L., Jeong H., Neda Z., Ravasz E., Schubert A., Vicsek T., Evolution of the social network of scientific collaborations. *Physica A*, **311**, (3-4), pp 590–614.

[2003] Barabasi A-L., Bonabeau E., Scale-free networks. *Scientific American*, pp 50–59.

[2012] Barber D., Bayesian reasoning and machine learning. Cambridge University Press.

[1998] Barberis N., Shleifer A., Vishny R., A model of investor sentiment. *Journal of Financial Economics*, **49**, pp 307–343.

[2002] Barberis N., Thaler T., A survey of behavioral finance. NBER Working Paper, 9222.

[1954] Barricelli N.A., Esempi numerici di processi di evoluzione. *Methodos*, pp 45–68.

[1988] Barron A.R., Barron R.L., Statistical learning networks: A unifying view. In E. Wegman, D. Gantz, and J. Miller, eds. *20th Symposium on the Interface: Computing Science and Statistics*, (American Statistical Association, Reston, Virginia) pp 192–203.

[1989] Barron A.R., Statistical properties of artificial neural networks. *Proceedings of the IEEE International Conference on Decision and Control*, pp 280–285, New York.

[1993] Barron A.R., Universal approximation bounds for superposition of a sigmoidal function. *IEEE Transactions on Information Theory*, **39**, pp 930–945.

[1994] Barron A.R., Approximation and estimation bounds for artificial neural networks. *Machine Learning*, **14**, (1), pp 115–133.

[1982] Barto A.G., Sutton R.S., Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element. *Behavioral Brain Research*, **4**, pp 221–235.

[1983] Barto A.G., Sutton R.S., Anderson C.W., Neuronlike elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics* 13, pp 835–846.

[1990] Barto A.G., Connectionist learning for control. In W.T. Miller III, R.S. Sutton, and P.J. Werbos, editors, *Neural Networks for Control*, chapter 1, pp 5–58. MIT Press, Cambridge.

[2012] Barunik J., Understanding the source of multifractality in financial markets. *Physica A*, **391**, (17), pp 4234–4251.

[2016] Bast H., Buchhold B., Haussmann E., Semantic search on text and knowledge bases. *Foundations and Trends in Information Retrieval*, **10**, (2-3), pp 119–271.

[2013] Battula B.P., Satya Prasad R., An overview of recent machine learning strategies in data mining. *International Journal of Advanced Computer Science and Applications*, **4**, (3), pp 50–54.

[1966] Baum L.E., Petrie T., Statistical inference for probabilistic functions of finite state Markov chains. *The annals of mathematical statistics*, **37**, (6), pp 1554–1563.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1970] Baum L.E., Petrie T., Soules G., Weiss N., A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, **41**, (1), pp 164–171.

[1972] Baum L.E., An inequality and associated maximization in statistical estimation for probabilistic functions of Markov processes. In *Proceedings of the 3rd Symposium on Inequalities, University of California, Los Angeles*.

[2001] Baxter J., Bartlett P., Infinite-horizon policy gradient estimation. *JAIR*, **15**, 319–350.

[2015] Baydin A.G., Pearlmutter B.A., Radul A.A., Siskind J.M., Automatic differentiation in machine learning: A survey. Working Paper, arXiv:1502.05767.

[2018] Bayer C., Stemper B., Deep calibration of rough stochastic volatility models. Preprint, arXiv:1810.03399.

[1993] Bazaraa M.S., Sherali H.D., Shetty C.M., Nonlinear programming theory and applications. 2nd ed. John Wiley and Sons.

[2003] Beal M.J., Variational algorithms for approximate Bayesian inference. PhD. Thesis, Gatsby Computational Neuroscience Unit, University College London.

[2015] Beauchemin M., Airflow: a workflow management platform. Medium. Archived.

[1992] Beer R.D., Gallagher J.C., Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, **1**, (1), pp 91–122.

[2008] Bekiros S.D., Georgoutsos D.A., Direction-of-change forecasting using a volatility based recurrent neural network. *Journal of Forecasting*, **27**, (5), pp 407–417.

[1992] Belew R.K., McInerney J., Schraudolph C.M., Evolving networks: Using the genetic algorithm with connectionist learning. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, pp 511–547. Addison-Wesley, Redwood City, CA.

[1995] Belin T.R., Rubin D.B., A method for calibrating false-match rates in record linkage. *Journal of the American Statistical Association*, **90**, (430), pp 694–707.

[2019] Belinkov Y., Glass J., Analysis methods in neural language processing: A survey. Transactions of the Association for Computational Linguistics, arXiv:1812.08951v2.

[1999] Bellgard C., Goldschmidt P., Forecasting foreign exchange rates: Random walk hypothesis, linearity and data frequency. Working Paper, The University of Western Australia.

[1954] Bellman R., The theory of dynamic programming. *Bull. Amer. Math. Soc.*, **60**, (5), pp 503–516.

[1957] Bellman R., Dynamic programming. Princeton University Press, Princeton.

[2017] Bello I., Pham H., Le Q.V., Norouzi M., Bengio S., Neural combinatorial optimizastion with reinforcement learning. in a conference paper at ICLR 2017.

[2007] Ben-Ameur H., Breton M., Karoui L., L'Ecuyer P., A dynamic programming approach for pricing options embedded in bonds. *Journal of Economic Dynamics and Control*, **31**, (7), pp 2212–2233.

[2003] Bender E.A., An introduction to mathematical modeling. Dover Publications Inc. First published in 1978.

[1994] Bengio Y., Simard P., Frasconi P., Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, **5**, (2), pp 157–166.

[2003] Bengio Y., Ducharme R., Vincent P., Janvin C., A neural probabilistic language model. *J. Mach. Learn. Res.*, **3**, pp 1137–1155.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2009] Bengio Y., Learning deep architectures for AI. Now Publishers Inc.

[2012] Bengio Y., Practical recommendations for gradient-based training of deep architectures. In Neural Networks: Tricks of the Trade. Springer.

[2013] Bengio Y., Courville A., Vincent P., Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, **35**, (8), pp 1798–1828.

[2001] Benhamou E., Delta equivalent asset flows. Swaps Strategy, Goldman Sachs International, London.

[2004] Bennell J., Sutcliffe C., Black-Scholes versus artificial neural networks in pricing FTSE 100 options. *Intelligent Systems in Accounting*, Finance and Management, **12**, pp 243–260.

[1996] Bennett F.H., Automatic creation of an efficient multi-agent architecture using genetic programming with architecture-altering operations. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp 30–38, Stanford University, CA, USA. MIT Press.

[1999] Bentley P., Kumar S., Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pp 35–43, Orlando, Florida, USA. Morgan Kaufmann.

[1994] Beran J., Statistics for long-memory processes. Monographs on Statistics and Applied Probability, Chapman & Hall, New York.

[2009] Bernardo J.M., Smith A.F.M., Bayesian Theory. Chichester, Wiley.

[1991] Bertsekas D.P., Tsitsiklis J.N., An analysis of the shortest stochastic path problems. *Mathematics of Operations Research*, **16**, pp 580–595.

[1996] Bertsekas D.P., Constrained optimization and Lagrange multiplier methods. MIT, Athena Scientific.

[1996] Bertsekas D.P., Tsitsiklis J.N., Neuro-dynamic programming. Athena.

[2005] Bertsekas D.P., Dynamic programming and optimal control. volume 1. Third Edition, Athena Scientific.

[2015] Bertsekas D.P., Convex optimization algorithms. Belmont, MA., Athena Scientific.

[1998] Bessembinder H., Chan K., Market efficiency and the returns to technical analysis. *Financial Management*, **27**, (2), pp 5–27.

[2009] Beynon-Davies P., Business information systems. Basingstoke, UK

[2012] Bhandari D., Murthy C.A., Pal S.K., Variance as a stopping criterion for genetic algorithms with elitist model. *Fundamenta Informaticae*, **120**, pp 145–164.

[2018] Bhandari D., Russo D., Singal R., A finite time analysis of temporal difference learning with linear function approximation. arXiv:1806.02450.

[2015] Bhardwaj A., Tiwari A., Breast cancer using genetically optimized neural network model. *International journal Expert System with Application*.

[2007] Bhatnagar S., Sutton R.S., Ghavamzadeh M., Lee M., Incremental natural actor-critic algorithms. In *Neural Information Processing Systems 21*.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2005] Bhattacharya I., Getoor L., Latent Dirichlet allocation model for entity resolution. Technical Report CS-TR-4740, Computer Science Dept., Univ. of Maryland.

[2017] Bianchi F.M., Kampffmeyer M., Maiorino E., Jensen R., Temporal overdrive recurrent neural network. Working Paper, University of Tromso, arXiv:1701.05159v1.

[1986] Biggs N., Lloyd E., Wilson R., Graph theory. Oxford University Press.

[2002] Bilenko M., Mooney R.J., Learning to combine trained distance metrics for duplicate detection in databases. Technical Report AI 02-296, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, TX, Feb.

[2003] Bilenko M., Mooney R.J., Cohen W.W., Ravikumar P., Fienberg S.E., Adaptive name matching in information integration. *IEEE Intelligent Systems*, **18**, (5), pp 16–23.

[2005] Bilenko M., Basu S., Sahami M., Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM-2005)*.

[1978] Binder D.A., Bayesian cluster analysis. *Biometrika*, **65**, pp 31–38.

[1931] Birkhoff G.D., Proof of the ergodic theorem. *Proc. Natl. Acad. Sci. USA*, **17**, (12), pp 656–660.

[1994] Bishop C.M., Mixture density networks. Technical Report, NCRG/94/004, Neural Computing Research Group, Aston University.

[2006] Bishop C.M., Pattern recognition and machine learning. Springer Verlag.

[2007] Bishop C.M., Lasserre J., Generative or discriminative? Getting the best of both worlds. In Bernardo, J.M., Bayesian statistics 8: proceedings of the eighth Valencia International Meeting, June 2-6, 2006, Oxford University Press, pp 3–23.

[1973] Bismut J.M., Conjugate convex functions in optimal stochastic control. *Journal Math. Anal. Appl.*, **44**, pp 384–404.

[2001] Blacher G., A new approach for designing and calibrating stochastic volatility models for optimal delta-vega hedging of exotic options. Conference presentation at Global Derivatives and Risk Management, Juan-les-Pins.

[1973] Black F., Scholes M., The pricing of options and corporate liabilities. *Journal of Political Economics*, **81**, pp 637–659.

[2003] Blei D.M., Ng A.Y., Jordan M.I., Latent Dirichlet allocation. *Journal of Machine Learning Research*, **3**, pp 993–1022.

[1996] Blickle T., Theory of evolutionary algorithms and application to system synthesis. PhD thesis, Swiss Federal Institute of Technology, Zurich.

[2010a] Bloch D.A., A note on calibration of Markov processes. *Wilmott Journal*, **2**, (2), pp 66–96.

[2010] Bloch D.A., A practical guide to implied and local volatility. Working Paper, University of Paris 6, SSRN-id1538808.

[2011] Bloch D.A., Coello Coello C.A., Smiling at evolution. *Applied Soft Computing*, **11**, (8), pp 5724–5734.

[2012] Bloch D., From implied to local volatility surface. Working Paper, University of Paris 6 Pierre et Marie Curie.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2014] Bloch D., A practical guide to quantitative portfolio trading. Working Paper, University of Paris 6 Pierre et Marie Curie.

[2019] Bloch D., Recipe for quantitative trading with machine learning. Working Paper, University of Paris 6 Pierre et Marie Curie.

[1954] Blum J.R., Approximation methods which converge with probability one. *The Annals of Mathematical Statistical Statistics*, **25**, (2), pp 382–386.

[2008] Boden M.A., Mind as machine: A history of cognitive science. Clarendon Press.

[2016] Bojanowski P., Grave E., Joulin A., Mikolov T., Enriching word vectors with subword information. arXiv:1607.04606.

[1986] Bollerslev T., Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, **31**, pp 307–327.

[1999] Bollerslev T., Jubinski D., Equality trading volume and volatility: Latent information arrivals and common long-run dependencies. *Journal of Business & Economic Statistics*, **17**, pp 9–21.

[1992] Bollinger J., Using Bollinger bands. *Technical Analysis of Stocks and Commodities*, **10**, February.

[1985] Bollobas B., Random graphs. Academic Press Inc, London.

[1991] Bollobas B., Probabilistic combinatorics and its applications. in *Proceeding of Symposia in Applied Mathematics*, **44**, 196 pp.

[2004] Bollobas B., Riordan O., The diameter of a scale-free random graph. *Combinatorica*, **24**, (1), pp 5–34.

[2003] Bondarenko O., Estimation of risk-neutral densities using positive convolution approximation. *Journal of Econometrics*, **116**, pp 85–112.

[2008] Bondy J.A., Murty U.S.R., Graph theory. Springer

[2001] Bongard J.C., Pfeifer R., Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, (GECCO-2001), pp 829–836, San Francisco, California, USA. Morgan Kaufmann.

[2003] Bongard J.C., Incremental approaches to the combined evolution of a robot's body and brain. PhD thesis, University of Zurich.

[2003] Bongard J.C., Pfeifer R., Evolving complete agents using artificial ontogeny. *Morpho-functional machines: The new species (designing embodied intelligence)*, pp 237–258.

[1847/2016] Boole G., The mathematical analysis of logic: Being an essay towards a calculus of deductive reasoning. Forgotten Books

[1854/2003] Boole G., The laws of thought. facsimile of 1854 edition, with an introduction by J. Corcoran. Buffalo: Prometheus Books (2003). Reviewed by James van Evra in Philosophy in Review, 24 (2004) pp 167–169.

[1998] Borodin A., El-Yaniv R., Online computation and competitive analysis. Cambridge University Press

[1992] Boser B.E., Guyon I.M., Vapnik V.N., A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pp 144.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2017] Bouizi R., Financial time series forecasting using wavelet transform and reservoir computing paradigm. Quant Finance Ltd, MSc Thesis, Department of Mathematics, Imperial College London.

[2012] Boulanger-Lewandowski N., Bengio Y., Vincent P., Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proceedings of the Twenty-nine International Conference on Machine Learning*, (ICML 12).

[1998] Bourbakis N.G., Artificial intelligence and automation. World Scientific.

[2014] Bowman G.R., Pande V.S., Noe F., An introduction to Markov state models and their application to long timescale molecular simulation. Advances in Experimental Medicine and Biology.

[2015] Bowman S.R., Angeli G., Potts C., Manning C.D., A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, (EMNLP).

[2004] Boyd S., Vandenberghe L., Convex optimization. Cambridge University Press.

[2000] Brabazon T., A connectivist approach to index modelling in financial markets. Department of Accountancy, University College, Dublin.

[2007] Bradley J., Schapire R., Filterboost: Regression and classification on large datasets. In Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems (NIPS).

[2001] Brameier M., Banzhaf W., A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation*, **5**, (1), pp 17–26.

[1995] Branicky M.S., Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, **138**, pp 67–100.

[1998] Breidt F.J., Crato N., de Lima P., On the detection and estimation of long memory in stochastic volatility. *Journal of Econometrics*, **83**, pp 325–348.

[1984] Breiman L., Friedman J., Stone C.J., Olshen R.A., Classification and regression trees. Chapman and Hall/CRC, Boca Raton, FL.

[1996] Breiman L., Stacked regressions. *Machine Learning*, **24**, (1), pp 49–64.

[1996b] Breiman L., Out-of-bag estimation. Technical report, Department of Statistics, University of California.

[1996c] Breiman L., Bagging predictors. *Machine Learning*, **24**, (2), pp 123–140.

[1996d] Breiman L., Bias, variance, and arcing classifiers. Statistics Department, University of California, Berkeley, CA.

[1996e] Breiman L., The heuristics of instability in model selection, in press. Annals of Statistics.

[2001] Breiman L., Random forests. *Machine learning*, **45**, (1), pp 5–32

[2006] Brest J., Greiner S., Boskovic B., Mernik M., Zumer V., Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, **10**, (6), pp 646–657.

[1992] Brock W., Lakonishok J., LeBaron B., Simple technical trading rules and the stochastic properties of stock returns. *Journal of Finance*, **47**, (5), pp 1731–1764.

[1991] Brockwell P.J., Davis R.A., Time series: Theory and methods. Springer Series in Statistics.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2018] Brostrom A., Kristiansson R., Exotic derivatives and deep learning. Degree Project in Mathematics, KTH Royal Institute of Technology, Stockholm, Sweden.

[1992] Brown P.F., Desouza P.V., Mercer R.L., Della Pietra V.J., Lai J.C., Class-based n-gram models of natural language. *Computational Linguistics*, **18**, (4), pp 467–479.

[1999] Brown C., Technical analysis for the trading professional. 1st Edition, McGraw-Hill.

[2020] Brown T.B., Mann B., Ryder N., Subbiah M., etc. Language models are few-shot learners. Working Paper, arXiv:2005.14165v4.

[1969] Bryson A.E., Ho Y.C., Applied optimal control. U.K. Blaisdell.

[2015] Bush N.T., Hasegawa H., Training artificial neural network using modification of differential evolution algorithm. *International Journal of Machine Learning and Computing*, **5**, (1), pp 1–6.

[2011] Bull J.J., Heineman R.H., Wilke C.O., The phenotype-fitness map in experimental evolution of phages. PLoS One, 6(11).

[2020] Bulusu S., Kailkhura B., Li B., Varshney P.K., Song D., Anomalous example detection in deep learning: A survey. ACCESS.2020.3010274, IEEE Access.

[2005] Bush K., Tsendjav B., Improving the richness of echo state features using next ascent local search. in *Proceedings of the Artificial Neural Networks in Engineering Conference*, pp 227–232, St. Louis.

[2006] Bush K., Anderson C., Exploiting iso-error pathways in the N, k-plane to improve echo state network performance.

[1994] Cai J., A Markov model of switching-regime ARCH. *Journal of Business*, **12**, pp 309–316.

[2006] Cai W., Pacheco-Vega A., Sen M., Yang K.T., Heat transfer correlations by symbolic regression. *International Journal of Heat and Mass Transfer*, **49**, (23-24), pp 4352–4359.

[2004] Cajueiro D.O., Tabak B.M., The Hurst exponent over time: Testing the assertion that emerging markets are becoming more efficient. *Physica A*, **336**, pp 521–537.

[2007] Cajueiro D.O., Tabak B.M., Long-range dependence and multifractality in the term structure of LIBOR interest rates. *Physica A*, **373**, pp 603–614.

[1995] Caldwell R.B., Performances metrics for neural network-based trading system development. *NeuroVet Journal*, **3**, (2), pp 22–26.

[2001] Calvet L., Fisher A., Forecasting multifractal volatility. *Journal of Econometrics*, **105**, pp 27–58.

[2002] Calvet L., Fisher A., Multifractality in asset returns: Theory and evidence. *The Review of Economics and Statistics*, **84**, (3), pp 381–406.

[2004] Calvet L., Fisher A., Regime-switching and the estimation of multifractal processes. *Journal of Financial Econometrics*, **2**, pp 44–83.

[2006] Calvet L., Fisher A., Thompson S., Volatility comovement: A multi-frequency approach. *Journal of Econometrics*, **31**, pp 179–215.

[2013] Calvet L., Fisher A., Wu L., Staying on top of the curve: A cascade model of term structure dynamics. Working Paper.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1997] Campbell J.Y., Lo A.W., MacKinlay A.C., The econometrics of financial markets. Princeton University Press, New jersey.

[2007] Candes E., Tao T., The dantzig selector: Statistical estimation when p is much larger than n. *The Annals of Statistics*, **35**, (6), pp 2313–2351.

[2008] Cao M., Qiao P., Neural network committee-based sensitivity analysis strategy for geotechnical engineering problems. *Neural Comput Appl*, **17**, pp 509–519

[2004] Carbone A., Castelli G., Stanley H., Time-dependent Hurst exponents in financial time series. *Physica A*, **344**, pp 267–271.

[2007] Carbone A., Algorithm to estimate the Hurst exponent of high-dimensional fractals. Working Paper, Physics Department, Politecnico di Torino.

[2014] Carr J., An introduction to genetic algorithms. Working Paper.

[1996] Casey M., The dynamics of discrete-time computation with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, **8**, pp 1135–1178.

[2001] Castiglione F., Bernaschi M., Market fluctuations: Simulation and forecasting. Working Paper

[1943] Cattell R.B., The description of personality: Basic traits resolved into clusters. *Journal of Abnormal and Social Psychology*, **38**, (4), pp 476–506.

[1857] Cayley A., On the theory of the analytical forms called trees. Philosophical Magazine, Series IV, **13**, (85), pp 172–176.

[2017] Cer D., Diab M., Agirre E., Lopez-Gazpio I., Specia L., SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp 1–14, Vancouver, Canada.

[2018] Cer D., Yang Y., Kong S-Y., Hua N., Limtiaco N., St.John N., etc Universal sentence encoder. arXiv preprint arXiv:1803.11175.

[2006] Cesa-Bianchi N., Lugosi G., Prediction, learning, and games. Cambridge University Press.

[2014] Chalamalla A., Ilyas I.F., Ouzzani M., Papotti P., Descriptive and prescriptive data cleaning. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, New York, NY, ACM, pp 445–456.

[2009] Chan E., Quantitative trading: How to build your own algorithmic trading business. John Wiley & Sons, volume 430.

[2008] Chance P., Learning and behavior. Belmont, CA, Wadsworth.

[1994] Chande T.S., Kroll S., The new technical trader. John Wiley, New York.

[2009] Chandola V., Banerjee A., Kumar V., Anomaly detection: A survey. *ACM Computing Surveys*, **41**, (3), pp 1–58.

[2010] Chapelle O., Scholkopf B., Zien A., Semi-supervised learning. *Adaptive computation and machine learning*, MIT Press.

[2018] Che Z., Purushotham S., Cho K., Sontag D.A., Liu Y., Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2014] Cheah Y-W., Quality, retrieval and analysis of provenance in large-scale data. Ph.D. dissertation, Indianapolis, Indiana University.

[1996] Cheeseman P., Sturz J., Bayesian classification (Autoclass): Theory and results. *Advances in Knowledge Discovery and Data Mining*, pp 153–180, AAAI Press/The MIT Press.

[1999] Chen S-H., Wang H-S., Zhang B-T., Forecasting high-frequency financial time series with evolutionary neural trees: The case of hengsheng stock index. In Hamid R. Arabnia, editor, *Proceedings of the International Conference on Artificial Intelligence, IC-AI 99*, volume 2, pp 437–443, Las Vegas, Nevada, USA. CSREA Press.

[2004] Chen Y., Yang B., Dong J., Nonlinear system modelling via optimal design of neural trees. *International Journal of Neural Systems*, **14**, (2), pp 125–137.

[2005] Chen S-H., Liao C-C., Agent-based computational modeling of the stock price-volume relation. *Information Sciences*, **170**, (1), pp 75–100.

[2009] Chen C-W., Huang C-S., Lai H-W., The impact of data snooping on the testing of technical analysis: An empirical study of Asian stock markets. *Journal of Asian Economics*, **14**, (5), pp 580–591.

[2016] Chen Y., Hoffman M.W., Colmenarejo S.G., Denil M., Lillicrap T.P., de Freitas N., Learning to learn for global optimization of black box functions. Working Paper.

[2017] Chen Q., Zhu X., Ling Z., Wei S., Jiang H., Inkpen D., Enhanced LSTM for natural language inference. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

[2007] Cheridito P., Soner H.M., Touzi N., Victoir N., Second-order backward stochastic differential equations and fully nonlinear parabolic pdes. *Communication on Pure and Applied Mathematics*, **60**, pp 1081–1110.

[1989] Chhabra A., Jensen R.V., Direct determination of the $f(\alpha)$ singularity spectrum. *Phys. Rev. Lett.*, **62**, (12), pp 1327–1330.

[2004] Ching W-K., Fung E.S., Ng M.K., Building higher-order Markov chain models with Excel. *International Journal of Mathematical Education in Science and Engineering*, **35**, pp 921–932.

[2004] Ching W-K., Zhu D-M., High-dimensional Markov chain models and their applications. Working Paper, University of Hong Kong.

[2010] Chitra A., Uma S., An ensemble model of multiple classifiers for time series prediction. *International Journal of Computer Theory and Engineering*, **2**, (3), pp 1793–8201

[2014] Cho K., Van Merrienboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk H., Bengio Y., Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv:1406.1078.

[2014b] Cho K., Van Merrienboer B., Bahdanau D., Bengio Y., On the properties of neural machine translation: Encoder-decoder approaches. Syntax, Semantics and Structure in Statistical Translation.

[2017] Choe Y.J., Shin J., Spencer N., Probabilistic interpretations of recurrent neural networks. Working Paper, Carnegie Mellon University.

[2019] Choi J., Kim T., Lee S-G., A cross-sentence latent variable model for semi-supervised text sequence matching. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp 4747–4761, Florence, Italy

[2017] Chollet F., Deep learning with python. New York, NY: Manning Publications.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1984] Chua L.O., Lin G.N., Non-linear programming without computation. *IEEE Trans. Circuits and Systems*, CAS-31, pp 182–186.

[2015] Chung J., Gulcehre C., Cho K., Bengio Y., Gated feedback recurrent neural networks. CoRR, abs:1502.02367.

[1993] Cichocki A., Unbehauen R., Neural networks for optimization and signal processing. John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore.

[2010] Ciresan D.C., Meier U., Gambardella L.M., Schmidhuber J., Deep big simple neural nets for handwritten digit recognition. *Neural Computation*, **22**, (12), pp 3207–3220.

[2011] Ciresan D.C., Meier U., Masci J., Gambardella L.M., Schmidhuber J., Flexible, high Performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, Volume Volume Two, (2), pp 1237–1242.

[2012] Ciresan D.C., Meier U., Schmidhuber J., Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, New York, pp 3642–3649.

[1983] Clarke F.H., Optimization and nonsmooth analysis. John Wiley & Sons, New York.

[2006] Clegg R.G., A practical guide to measuring the Hurst parameter. *International Journal of Simulation: Systems, Science and Technology*, **7**, (2), pp 3–14.

[2015] Clevert D-A., Unterthiner T., Hochreiter S., Fast and accurate deep network learning by exponential linear units (elus). Working Paper, ArXiv e-prints.

[2009] Clune J., Beckmann B.E., Pennock R.T., Ofria C., Hybrid: A hybridization of indirect and direct encodings for evolutionary computation. In *Proceedings of the 10th European Conference on Artificial Life*, Budapest, Hungary.

[2013] Clune J., Mouret J-B., Lipson H., The evolutionary origins of modularity. In *Proceedings of the Royal Society B*, 280(20122863).

[2011] Coates A., Lee H., Ng A.Y., An analysis of single-layer networks in unsupervised feature learning. In *Int. Conf. on AI and Statistics*, (AISTATS).

[2001] Cochinwala M., Kurien V., Lalk G., Shasha D., Efficient data reconciliation. *Information Sciences*, **137**, (1-4), pp 1–15.

[1970] Codd E.F., A relational model of data for large shared data banks. *Communications of the ACM*, **13**, (6), pp 377–387.

[1971] Codd E.F., Further normalization of the data base relational model. (Presented at Courant Computer Science Symposia Series 6, Data Base Systems, New York City, May 24-25, 1971.) IBM Research Report RJ909 (August 31, 1971). Republished in Randall J. Rustin (ed.), Data Base Systems: Courant Computer Science Symposia Series 6. Prentice-Hall, 1972.

[2000] Coello Coello C.A., Constraint-handling using an evolutionary multiobjective optimization technique. *Civil Engineering and Environmental Systems*, Vol. **17**, pp 319–346.

[2001] Coello Coello C.A., Pulido G.T., A Micro-Genetic Algorithm for multiobjective optimization. in International Conference on Evolutionary Multi-Criterion Optimization EMO 2001, Evolutionary Multi-Criterion Optimization, pp 126–140.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2002] Coello Coello C.A., Mezura-Montes E., Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatic*, Vol. **16**, pp 193–203.

[2003] Coello Coello C.A., Mezura-Montes E., Increasing successful offspring and diversity in differential evolution for engineering design. *Advanced Engineering Informatic*, Vol. **16**, pp 193–203.

[1960] Cohen J., A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, **20**, (1), pp 37–46.

[1998] Cohen W.W., Integration of Heterogeneous Databases without Common Domains Using Queries Based on Textual Similarity. In *Proc. 1998 ACM SIGMOD Int'l Conf. Management of Data (SIGMOD'98)*, pp 201–212.

[2000] Cohen R., Erez K., ben-Avraham D., Havlin S., Resilience of the internet to random breakdowns. *Physical Review Letters*, **85**, (21), pp 4626–4628.

[2000b] Cohen W.W., Kautz H., McAllester D., Hardening soft information sources. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD-2000)*, pp 255–259, Boston, MA.

[2002] Cohen W.W., Richman J., Learning to match and cluster large high-dimensional data sets for data integration. In *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '02)*.

[2003] Cohen W.W., Ravikumar P., Fienberg S., A comparison of string distance metrics for name-matching tasks. In *Proceedings of KDD Workshop on Data Cleaning and Object Consolidation, AAAI*.

[2014] Colledanchise M., Ogren P., How behavior trees modularize robustness and safety in hybrid systems. In *Intelligent Robots and Systems (IROS)*, 2014 IEEE/RSJ International Conference on, IEEE.

[1997] Collins E., Finite-horizon variance penalised Markov decision processes. *OR Spektrum*, **19**, pp 35–39.

[2008] Collobert R., Weston J., A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pp 160–167.

[2011] Collobert R., Weston J., Bottou L., Karlen M., Kavukcuoglu K., Kuksa P., Natural language processing (almost) from scratch. *J. Machine Learning Research*, **11**, pp 2493–2537.

[2017] Conneau A., Kiela D., Schwenk H., Barrault L., Bordes A., Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp 670–680, Copenhagen, Denmark. Working Paper, arXiv:1705.02364v5.

[2018] Conneau A., Kiela D., SentEval: An evaluation toolkit for universal sentence representations. arXiv preprint arXiv:1803.05449.

[1994] Connor J.T., Martin R.D., Atlas L.E., Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, **5**, (2), pp 240–254.

[2002] Cont R., Tankov P., Calibration of jump-diffusion option-pricing models: A robust non-parametric approach. Ecole Polytechnique, CMAP, September.

[2016] Contras D., Matei O., Translation of the mutation operator from genetic algorithms to evolutionary ontologies. In *International Journal of Advanced Computer Science and Applications (IJACSA)*, **7**, (1).

[1990] Cooper G.F., The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, **42**, (2-3), pp 393–405.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2001] Cormen T,H., Leiserson C.E., Rivest R.L., Stein C., Introduction to algorithms. 2nd ed., MIT Press and McGraw-Hill. Third Edition is from 2009.

[2007] Cornuejols G., Revival of the Gomory cuts in the 1990s. *Annals of Operations Research*, **149**, pp 63–66.

[1995] Cortes C., Vapnik V., Support-vector networks. *Machine Learning*, **20**, (3), pp 273–297.

[2002] Cortez P., Rocha M., Neves J., A lamarckian approach for neural network training. *Neural Processing Letters*, **15**, pp 105–116.

[2013] Cortez P., Embrechts M.J., Using sensitivity analysis and visualization techniques to open black box data mining models. *Inf Sci*, **225**, pp 1–17

[2003] Costa R.L., Vasconcelos G.L., Long-range correlations and nonstationarity in the Brazilian stock market. *Physica A*, **329**, pp 231–248.

[2006] Cox D.R., Principles of statistical inference. Cambridge University Press.

[1985] Cramer N.L., A representation for the adaptive generation of simple sequential programs. In *Proceedings of an International Conference on Genetic Algorithms and the Applications*, Carnegie Mellon University, Pittsburgh, pp 183–187.

[2008] Cretu G.F., Stavrou A., Locasto M.E., Stolfo S.J., Keromytis A.D., Casting out demons: Sanitizing training data for anomaly sensors. In *IEEE S&P*, pp 81–95.

[2017] Culkin R., Das S.R., Machine learning in finance: The case of deep learning for option pricing. Working Paper, Santa Clara University.

[2015] Cully A., Clune J., Tarapore S.R., Mouret J-B., Robots that can adapt like animals. *Nature*, volume 521, pp 503–507.

[1990] Cun Y.L., Denker J., Solla S., Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing*, Systems 2. Morgan Kaufmann, San Mateo, CA.

[2000] Cunningham P., Carney J., Diversity versus quality in classification ensembles based on feature selection. Technical Report TCD-CS-2000-02, Department of Computer Science, Trinity College Dublin.

[2010] Curry H.B., Foundations of mathematical logic. Dover Publications Inc. First published in 1977.

[1988] Cybenko G., Continuous valued neural networks with two hidden layers are sufficient. Technical Report, Department of Computer Science, Tufts University, Medford, MA.

[1989] Cybenko G., Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, **2**, (4), pp 303–314.

[2003] Dablemont S., Van Bellegem S., Verleysen M., Modelling and forecasting financial time series of tick data by functional analysis and neural networks. Universite catholique de Louvain, Machine Learning Group, DICE, Louvain-la-Neuve, Belgium.

[1993] Dacorogna M.M., Muller U.A., Nagler R.J., Olsen R.B., Pictet O.V., A geographical model for the daily and weekly seasonal volatility in the foreign exchange market. *Journal of International Money and Finance*, **12**, (4), pp 413–438.

[1998] Dacorogna M.M., Muller U.A., Olsen R.B., Pictet O.V., Modelling short-term volatility with GARCH and HARCH models. in C. Dunis and B. Zhou, (eds.), *Nonlinear Modelling of High Frequency Financial Time Series*, John Wiley & Sons, pp 161–176.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2001] Dacorogna M.M., Gencay R., Muller U.A., Olsen R.B., Pictet O.V., An introduction to high frequency finance. Academic Press, San Diego, CA.

[2006] Daglish T., Hull J., Suo W., Volatility surfaces: Theory, rules of thumb and empirical evidence. Working Paper, August.

[1993] Dagum P., Luby M., Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, **60**, (1), pp 141–153.

[1997] Dagum P., Luby M., An optimal approximation algorithm for Bayesian inference. *Artificial Intelligence*, **93**, (1-2): pp 1–27.

[2015] Dai W., Yoshigoe K., Parsley W., Improving data quality through deep learning and statistical models. Computer Science, University of Arkansas at Little Rock.

[2017] Dai X., Bikdash M., Meyer B., From social media to public health surveillance: Word embedding based clustering method for twitter classification. In *SoutheastCon*, IEEE, 2017, pp 1–7.

[2017] Dalal G., Szorenyi B., Thoppe G., Mannor S., Finite sample analysis for $TD(0)$ with function approximation. arXiv:1704.01161. Also appeared in AAAI 2018.

[2007] D'Ambrosio D.B., Stanley K.O., A novel generative encoding for exploiting neural network sensor and output geometry. In *Genetic And Evolutionary Computation Conference: Proceedings of the 9th annual conference on Genetic and Evolutionary Computation*, Association for Computing Machinery, New York, NY.

[2011] Damelin S., Miller W., The mathematics of signal processing. Cambridge University Press

[2003] Damodaran A., Investment philosophies. John Wiley & Sons, New York.

[1994] Dana R.A., Jeanblanc-Picque M., Marche financiers en temps continu: valorisation et equilibre. Recherche en Gestion, Economica, Paris.

[2004] Dangalchev C., Generation models for scale-free networks. *Physica A: Statistical Mechanics and its Applications*, **338**, (3-4), pp 659–671.

[1998] Darbellay G.A., Predictability: An information-theoretic perspective. in *Signal Analysis and Prediction*, edt, A. Prochazka, J. Uhlir, P.W.J. Rayner and N.G. Kingsbury, pp 249–262.

[2000] Darbellay G.A., Slama M., Forecasting the short-term demand for electricity? Do neural networks stand a better chance? *International Journal of Forecasting*, **16**, pp 71–83.

[1882] Darwin C.R., The variation of animals and plants under domestication. Murray, London, second edition.

[2005] Das S., Konar A., Chakraborty U.K., Two improved differential evolution schemes for faster global search. in *Proceedings of the 2005 conference on Genetic and Evolutionary Computing*, (GECCO 2005), pp 991–998.

[2017] Dauphin Y.N., Fan A., Auli M., Grangier D., Language modeling with gated convolutional networks. arXiv:1612.08083.

[2005] Davies E.R., Machine vision: Theory, algorithms, practicalities. Morgan Kaufmann.

[1973] Davis M.H.A., On the existence of optimal policies in stochastic control. *SIAM J. Control Optim.*, **11**, pp 587–594.

[1993] Davis M.H.A., Markov models and optimization. Chapman & Hall, London.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1993] Davis R., Shrobe H., Szolovits P., What Is a Knowledge Representation? *AI Magazine*, **14**, (1), pp 17–33.

[2015] Davis E., Knowledge Representation In International Encyclopedia of the Social & Behavioral Sciences (Second Edition).

[1992] Dayan P., The convergence of $TD(\lambda)$ for general λ. *Machine Learning*, **8**, pp 341–362.

[2002] Day T.E., Wang P., Dividends, nonsynchronous prices, and the returns from trading the Dow Jones Industrial Average. *Journal of Empirical Finance*, **9**, (4), pp 431–454.

[2004] Dean J., Ghemawat S., MapReduce: Simplified data processing on large clusters. Google.

[2015] Dean J., Monga R., TensorFlow: Large-scale machine learning on heterogeneous systems. Google Research.

[1989] Deb K., Goldberg D.E., An investigation of niche and species formation in genetic function optimization. In J.D. Schaffer, Ed., Proceedings of the Third International Conference on Genetic Algorithms, pp 42–50, San Mateo, CA. Morgan Kaufmann.

[2000] Deb K., An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, **186**, (2/4), pp 311-338.

[2001] Deb K., Multi-objective optimization using evolutionary algorithms. Wiley.

[2013] de Carvalho M.G., Laender A.H.F., Goncalves M.A., da Silva A.S., A genetic programming approach to record deduplication. *IEEE Transaction on Knowledge and Data Engineering*, pp 399–412.

[2013] Deepa K., Rangarajan R., Selvi M.S., Automatic threshold selection using PSO for GA based duplicate record detection. *International Journal of Computer Applications*, **62**, (4).

[1990] Deerwester S., Dumais S., Laudauer T., Furnas G., Harshman R., Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, **41**, (6), pp 391–407.

[2010] de Freitas J., Pappa G.L., da Silva A.S., Goncalves M.A., Moura E., Veloso A., Laender A.H.F., de Carvalho M.G., Active learning genetic programming for record deduplication. *Evolutionary Computation (CEC)*, 2010 IEEE Congress, pp 1–8.

[2014] Degirmenci A., Introduction to hidden Markov models. Lecture Notes.

[2012] Degris T., Pilarski P.M., Sutton R.S., Model-free reinforcement learning with continuous action in practice. In *American Control Conference*.

[2012b] Degris T., White M., Sutton R.S., Linear off-policy actor-critic. In *29th International Conference on Machine Learning*.

[1977] Dempster A.P., Laird N.M., Rubin D.B., Maximum likelihood from incomplete data via the em algorithm. In *Journal of the Royal Statistical Society*.

[2000] Dempster M.A.H., Jones C.M., A real-time adaptive trading system using genetic programming. *Quantitative Finance*, **1**, pp 397–413.

[2001] Dempster M.A.H., Payne T.W., Romahi Y., Thompson G.W.P., Computational learning techniques for intraday FX trading using popular technical indicators. *IEEE Transactions on Neural Networks*, **12**, (4), pp 744–754.

[2014] Deng L., Yu D., Deep learning: Methods and applications. Now Publishers

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1989] Denker J.S., Gardner W.R., Graf H.P., Henderson D., Howard R.E., Hubbard W., Jackel L.D., Baird H.S., Guyon I., Neural network recognizer for hand-written zip code digits. AT&T Bell Laboratories.

[2014] De Ona J., Garrido C., Extracting the contribution of independent variables in neural network models: A new approach to handle instability. *Neural Computing and Applications*, **25**, pp 859–869.

[1999] Derman E., Regimes of volatility. *Risk*, **7**, (2).

[2008] Derman E., Lecture Notes 1–12.

[2018] de Sousa Marinho A.V., Approximate string matching and duplicate detection in the deep learning era. MSc thesis in Computer Engineering, Tecnico, Lisbon.

[2018] Devlin J., Chang M-W., Lee K., Toutanova K., Bert: Pre-training of deep bidirectional transformers for language understanding. Working Paper, arXiv:1810.04805. In *Proceedings of 2019 NAACL*.

[2007] DeVor R.E., Chang T-H., Lee K., Sutherland J.W., Statistical quality design and control: Contemporary concepts and methods. Prentice Hall.

[1994] D'haeseleer P., Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pp 256–261, Orlando, Florida, USA. IEEE Press.

[1945] Dice L.R., Measures of the amount of ecologic association between species. *Ecology*, **26**, (3).

[1995] Dieckmann U., Coevolution as an autonomous learning strategy for neuromodules. In Hans J. Herrmann, D. E. Wolf, and Ernst Poppel, editors, Workshop on Supercomputing in Brain Research: from tomography to neural networks, HLRZ, KFA Julich, Germany, pp 21–23. World Scientific Publishing Co..

[2014] Dieleman S., Recommending music on Spotify with deep learning. Working Paper.

[2000] Dietterich T.G., An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, **40**, (2), pp 139–158.

[1993] Ding Z., Granger C.W.J., Engle R.F., A long memory property of stock market returns and a new model. *Journal of the Empirical Finance*, **1**, pp 83–106.

[2017] Dingli A., Fournier K.S., Financial time series forecasting: A machine learning approach. *Machine Learning and Applications: An International Journal* (MLAIJ), **4**, No.1/2/3, September.

[2016] Doersch C., Tutorial on variational autoencoders. arXiv:1606.05908.

[2000] Dominey P.F., Ramus F., Neural network processing of natural language: I. Sensitivity to serial, temporal and abstract structure of language in the infant. *Language and Cognitive Processes*, **15**, (1), pp 87–127.

[2014] Donahue J., Hendricks L., Guadarrama S., Rohrbach M., Venugopalan S., Saenko K., Darrell T., Long-term recurrent convolutional networks for visual recognition and description. CoRR, abs/1411.4389.

[2017] Donahue J., Krahenbuhl P., Darrell T., Adversarial feature learning. In *ICLR*.

[2003] Doncieux S., Meyer J-A., Evolving neural networks for the control of a lenticular blimp. Lecture notes in computer science, pp 626–637.

[2004a] Doncieux S., Meyer J-A., Evolving modular neural networks to solve challenging control problems. In *Proceedings of the Fourth International ICSC Symposium on engineering of intelligent systems*, Citeseer.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2004b] Doncieux S., Meyer J-A., Evolution of neurocontrollers for complex systems: Alternatives to the incremental approach. In *Proceedings of The International Conference on Artificial Intelligence and Applications*, Citeseer.

[2005] Doncieux S., Meyer J-A., Evolving pid-like neurocontrollers for nonlinear control problems. *Control and Intelligent Systems*, ACTA Press, **33**, (1), pp 55–62.

[2005] Dong X., Halevy A., Madhavan J., Reference reconciliation in complex information spaces. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data (SIGMOD-2005)*, pp 85–96, Baltimore, MD.

[2015] Dong X.L., Gabrilovich G., Murphy K., Dang V., Horn W., Lugaresi C., Sun S., Zhang W., Knowledge-based trust: Estimating the trustworthiness of web sources. *Proc. VLDB Endow.*, **8**, (9), pp 938–949.

[2002] Dokas P., Ertoz L., Kumar V., Lazarevic A., Srivastava J., Tan P-N., Data mining for network intrusion detection. In *Proceedings NSF Workshop on Next Generation Data Mining*.

[2000] Donoho D.L., High-dimensional data analysis: The curses and blessings of dimensionality. Invited lecture at Mathematical Challenges of the 21st Century, AMS National Meeting, Los Angeles, CA, USA, August pp 6–12.

[1937] Doob J.L., Stochastic processes depending on a continuous parameter. *Transactions of the American Mathematical Society*, **42**, (1), pp 107–140.

[1953] Doob J.L., Stochastic processes. John Wiley & Sons, New York.

[2018] Dor L.E., Mass Y., Halfon A., Venezian E., Shnayderman I., Aharonov R., Slonim N., Learning thematic similarity metric from article sections using triplet networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp 49–54, Melbourne, Australia. Association for Computational Linguistics.

[2000] Dorogovtsev S.N., Mendes J.F.F., Samukhin A.N., Structure of growing networks with preferential linking. *Phys. Rev. Lett.*, **85**, (21), 4633.

[2011] dos Santos J.B., Heuser C.A., Moreira V.P., Wives L.K., Automatic threshold estimation for data matching applications. *Information Sciences*, **181**, (13), pp 2685–2699.

[1999] Douglas S.C. Introduction to adaptive filters. University of Utah. Published by CRC Press LLC.

[2000] Dowd K., Adjusting for risk: An improved Sharpe ratio. *International Review of Economics and Finance*, **9**, (3), pp 209–222.

[2007] Downing K.L., Supplementing evolutionary developmental systems with abstract models of neurogenesis. In Dirk Thierens, Hans-Georg Beyer, Josh Bongard, Jurgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev Kumar, Julian F. Miller, Jason Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stutzle, Richard A Watson, and Ingo Wegener, editors, GECCO 07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, volume 1, pp 990–996, London. ACM Press.

[1992] Doya K., Bifurcations in the learning of recurrent neural networks. in *Proceedings of IEEE International Symposium on Circuits and Systems*, **6**, pp 2777–2780.

[1993] Doya K., Bifurcations of recurrent neural networks in gradient descent learning. *IEEE Transactions on Neural Networks*, **1**, pp 78–80.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1996] Drucker H., Burges C.J.C., Kaufman L., Smola A.J., Vapnik V.N., Support vector regression machines. in *Advances in Neural Information Processing Systems 9, (NIPS)*, MIT Press, pp 155–161.

[1988] Druckman D., Swets J.A., Enhancing human performance: Issues, theories, and techniques. National Academy Press.

[1999] Drummond C., Hearne T., The lessons. A series of 30 multi-media lessons, Drummond and Hearne Publications, Chicago.

[1969] Du Bois Jr N.S.D., A solution to the problem of linking multivariate documents. *J. Am. Statistical Assoc.*, **64**, (325), pp 163–174.

[2018] Dubois G., Modeling and simulation. Taylor & Francis, CRC Press.

[2011] Duchi J., Hazan E., Singer Y., Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, **12**, pp 2121–2159.

[1973] Duda R.O., Hart P.E., Pattern classification and scene analysis. Wiley.

[1983] Dudgeon D., Mersereau R., Multidimensional digital signal processing. Prentice-Hall.

[1992] Duffie D., Dynamic asset pricing theory. Princeton University Press.

[1992b] Duffie D., Epstein L., Stochastic differential utility. *Economitrica*, **60**, (2), pp 353–394.

[1992c] Duffie D., Epstein L., Asset pricing with stochastic differential utility. *Rev. Financial Stud.*, **5**, pp 411–436.

[1994] Dupire B., Pricing with a smile. *Risk*, **7**, pp 18–20.

[1987] Durbin R., Willshaw D., An analogue approach to the Travelling Salesman. *Nature*, **326**, (16), pp 689–691.

[2003] Durrleman V., A note on initial volatility surface. Working Paper.

[2006] Duval R., A cognitive analysis of problems of comprehension in a learning of mathematics. *Educational Studies in Mathematics*, **61**, No. 1, pp 103–131.

[2019] Dyakonov M., The case against quantum computing. IEEE Spectrum, March, 2019.

[1956] Dynkin E.B., Yushkevich A.A., Strong Markov Processes. *Teor. Veroyatnost. i Primenen.*, **1**, (1), pp 149–155, Theory Probab. Appl., **1**, (1), pp 134–139.

[1961] Dynkin E.B., Theory of Markov processes. (translated from the Russian by D.E. Brown, edited by T. Kovary). Prentice-Hall Inc., Englewood Cliffs, N.J. and Pergamon Press, Oxford-London-Paris.

[1965] Dynkin E.B., Markov processes, vol. I and II. Springer-Verlag.

[1979] Dynkin E.B., Yushkevich A.A., Controlled Markov processes. Springer-Verlag, New York, **235**.

[1953] Eccles J.G., The Neurophysiological basis of mind. Clarendon, Oxford.

[1995] Edmonds A.N., Burkhardt D., Adjei O., Genetic programming of fuzzy logic production rules with application to financial trading. Neural networks in Financial Engineering, World Scientific, pp 179–188.

[1993] Efron B., Tibshirani R., An introduction to the bootstrap. Chapman & Hall, New York.

[2003] Eiben A.E., Smith J.E., Introduction to evolutionary computing. Springer.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1908] Einstein A., Uber die von der molekularkinetischen theorie der warme geforderte bewegung von in ruhenden flussigkeiten suspendierten teilchen. *Annals of Physics*, **322**.

[2002] Elfeky M.G., Verykios V.S., Elmargarid A.K., Tailor: A record linkage toolbox. In *Proceedings of the 18th International Conference on Data Engineering (ICDE '02)*, pp 17, Washington, DC, USA. IEEE Computer Society.

[1995] El Hihi S., Bengio Y., Hierarchical recurrent neural networks for long-term dependencies. In *NIPS*, **400**, pp 409.

[1981] El Karoui N., Les aspects probabilistes du controle stochastique. *Lecture Notes in Math.*, **816**, Berlin and New York, Springer-Verlag, pp 73–238.

[1995] El Karoui N., Quenez M.C., Dynamic programming and pricing of contingent claims in an incomplete market. *SIAM Journal on Control Optim.*, **33**, pp 29–66. Preprints date from 1992 and 1993.

[1998] El Karoui N., Jeanblanc M., Shreve S., Robustness of the BS formula. *Mathematical Finance*, **8**, pp 93-126.

[2007] Elmagarmid A.K., Ipeirotis P.G., Verykios V.S., Duplicate Record Detection: A Survey. *IEEE Transactions On Knowledge And Data Engineering*, **19**, (1).

[1990] Elman J.L., Finding structure in time. *Cognitive Science*, **14**, (2).

[1993] Elman J.L., Learning and development in neural networks: The importance of starting small. *Cognition*, **48**, pp 71–99.

[1999] Embrechts P., Resnick S., Samorodnitsky G., Extreme value theory as a risk management tool. *North American Actuarial Journal*, **26**, pp 30-41.

[2002] Embrechts P., Maejima M., Selfsimilar processes. Princeton University Press.

[2003] Engels J.M., Diehr P., Imputation of missing longitudinal data: A comparison of methods. *Journal of Clinical Epidemiology*, **56**, pp 968–976.

[1982] Engle R.F., Autoregressive conditional heteroskedasticity with estimates of the variance of U.K. inflation. *Econometrica*, **50**, pp 987–1008.

[1986] Engle R.F., Granger C.W.j., Rice j., Weiss A., Semiparametric estimates of the relation between weather and electricity sales. *Journal of the American Statistical Association*, **81**, pp 310–320.

[1987] Engle R.F., Granger C.W.j., Co-integration and error correction: Representation, estimation, and testing. *Econometrica*, **55**, 2, pp 251–276.

[1959] Erdos P., Renyi A., On random graphs. In *Publ. Math*, pp 290–297.

[2010] Erhan D., Bengio Y., Courville A., Manzagol P-A., Vincent P., Bengio S., Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, **11**, (Feb), pp 625–660.

[2005] Erl T., Service-oriented architecture: Concepts, technology and design. Prentice Hall.

[2005] Ernst D., Geurts P., Wehenkel L., Tree-based batch model reinforcement learning. *Journal of Machine Learning Research*, **6**, pp 405–556.

[2010] Esuli A., Sebastiani F., Evaluating Information Extraction. In *Cross-Language Evaluation Forum (CLEF)*, pp 100–111.

[1736] Euler L., Solutio problematis ad geometriam situs pertinentis. *Comment. Acad. Sci. U. Petrop 8*, pp 128–40.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2011] Everitt B., Cluster analysis. Chichester, West Sussex, U.K.

[1965] Fama E.F., Portfolio analysis in a stable Paretian market. *Management Science*, **11**, pp 404–419.

[2003] Fan H.Y., Lampinen J., A trigonometric mutation operation to differential evolution. *International Journal of Global Optimization*, **27**, pp 105–129.

[1954] Farley B.G, Clark W.A., Simulation of self-organizing systems by digital computer. *IRE Transactions on Information Theory*, **4**, (4), pp 76–84.

[2014] Faruqui M., Dyer C., Improving vector space word representations using multilingual correlation. In *Proceedings of 2014 EACL*.

[2016] Faruqui M., Dodge J., Jauhar S.K., Dyer C., Hovy E., Smith N.A., Retrofitting word vectors to semantic lexicons. In *Proceedings of 2016 NAACL*.

[1994] Fausett L.V., Fundamentals of neural networks: architectures, algorithms, and applications. Prentice-Hall.

[2002] Favre L., Galeano J.A., Mean-modified value-at-risk optimization with hedge funds. *Journal of Alternative Investments*, **5** (Fall), pp 21–25.

[2006] Fawcett T., An introduction to ROC analysis. *Pattern Recognition Letters*, **27**, (8), pp 861–874

[1988] Feder J., Fractals. Plenum Press, New York.

[1969] Fellegi I.P., Sunter A.B., A theory for record linkage. *J. Am. Statistical Assoc.*, **64**, (328), pp 1183–1210.

[1951] Feller W., Two singular diffusion problems. *Annals of Mathematics*, **54**, pp 173–182.

[2005] Fengler M.R., Semiparametric modeling of implied volatility. Springer Finance, Springer-Verlag Berlin Heidelberg.

[2006] Feoktistov V., Differential evolution, in search of solutions. Springer, Optimisation and its Applications, **5**.

[2008] Fernandez-Blanco P., Technical market indicators optimization using evolutionary algorithms. In *Proceedings of the 2008 GECCO conference companion on genetic and evolutionary computation*. Atlanta, USA.

[2016] Fernando C., Banarse D., Reynolds M., Besse F., Pfau D., Jaderberg M., Lanctot M., Wierstra D., Convolution by evolution: Differentiable pattern producing networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, (GECCO 16). Denver, Colorado, pp 109–116.

[2012] Finch H., Distribution of variables by method of outlier detection. *Frontiers in Psychology*, **3**, pp 1–12.

[2005] Finkel J.R., Grenager T., Manning C., Incorporating non-local information into information extraction systems by Gibbs Sampling. In *43rd Annual Meeting of the Association for Computational Linguistics*, pp 363–370.

[1957] Firth J.R., A synopsis of linguistic theory 1930–1955. Studies in Linguistic Analysis, pp 1–32. Reprinted in F.R. Palmer, ed. (1968). Selected Papers of J.R. Firth 1952–1959, London, Longman.

[1936] Fisher R.A., The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, **7**, pp 179–188.

[1938] Fisher R.A., The statistical utilization of multiple measurements. *Annals of Eugenics*, **8**, pp 376–386.

[1981] Fleiss J.L., Statistical methods for rates and proportions. John Wiley & Sons, New York, NY, 2nd edition.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1975] Fleming W., Rishel R., Deterministic and stochastic control. Springer-Verlag, Berlin.

[2008] Floreano D., Durr P., Mattiussi C., Neuroevolution: From architectures to learning. *Evolutionary Intelligence*, **2**, (1), pp 47–62.

[1966] Fogel L.J., Artificial intelligence through simulated evolution. John Wiley, New York.

[2008] Fok W.W.T., Tam V.W.L., Computational neural network for global stock indexes prediction. In *Proceedings of the World Congress on Engineering*, **2**, pp 1171–1175.

[1986] Follmer H., Sondermann D., Hedging of non-redundant contingent claims. In W. Hilenbrand and A. Mas-Colell (eds.) *Contributions to Mathematical Economics*, pp 205–223.

[1989] Follmer H., Schweitzer M., Hedging by sequential regression: An introduction to the mathematics of option trading. *ASTIN Bulletin*, **18**, pp 147–160.

[1990] Follmer H., Schweizer M., Hedging of contingent claims under incomplete information. In M.H.A. Davis and R.J. Elliott, editors, *Applied Stochastic Analysis*, pp 389–414, Gordon and Breach, London.

[1990] Fornasini E., 2D Markov chains. Department of Electronics and Computer Science, University of Padova, Italy.

[1981] Forsyth R., BEAGLE a Darwinian approach to pattern recognition. *Kybernetes*, **10**, (3), pp 159–166.

[1988] Fort J.C., Solving a combinatorial problem via self-organizing process: An application of the Kohonen algorithm to the traveling salesman problem. *Biological Cybernetics*, **59**, (1), pp 33–40.

[2001] Foster J.A., Review: Discipulus: A commercial genetic programming system. *Genetic Programming and Evolvable Machines*, **2**, (2), pp 201–203.

[1986] Fox R., Taqqu M.S., Large-sample properties of parameter estimates for strongly dependent stationary Gaussian time series. *The Annals of Statistics*, **14**, (2), pp 517–532.

[1957] Fraser A., Simulation of genetic systems by automatic digital computers. I. Introduction. *Aust. J. Biol. Sci.*, **10**, (4), pp 484–491.

[2009] Freedman D.A., Statistical models. Cambridge University Press

[1995] Freund Y., Schapire R.E., A decision-theoretic generalization of on-line learning and an application to boosting. Working Paper.

[1996] Freund Y., Schapire R.E., Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pp 148–156.

[1997] Freund Y., Schapire R.E., A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, **55**, (1), pp 119–139.

[2001] Freund Y., An adaptive version of the boost by majority algorithm. *Machine Learning*, **43**, (3), pp 293–318.

[2005] Friedland B., Control system design: An Introduction to state-space methods. Dover.

[1998] Friedman J.H., Data mining and statistics: What's the connection? *Computing Science and Statistics*, **29**, (1), pp 3–9.

[2000] Friedman J., Hastie T., Tibshirani R., Additive logistic regression: A statistical view of boosting (with discussions). *Annals of Statistics*, **28**, (2), pp 337–407.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2001] Friedman J.H., Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, pp 1189–1232.

[2015] Frieze A., Karonski M., Introduction to random graphs. Cambridge University Press.

[1985] Frisch U., Parisi G., Fully developed turbulence and intermittency. in M. Ghil, R. Benzi, G. Parisi, ed., *Turbulence and Predictability in Geophysical Fluid Dynamics and Climate Dynamics*, Amsterdam, pp 84–88.

[2010a] Fruehwirth-Schnatter S., Fruehwirth R., Data augmentation and MCMC for binary and multinomial logit models. In T. Kneib and G. Tutz, Eds., *Statistical Modelling and Regression Structure*, Festschrift in Honour of Ludwig Fahrmeir, Heidelberg, Physica-Verlag, pp 111–132.

[2010b] Fruehwirth-Schnatter S., Wagner H., Stochastic model specification search for Gaussian and partially non-Gaussian state space models. *Journal of Econometrics*, **154**, pp 85–100.

[2005] Fu M.C., Glover F.W., April J., Simulation optimization: A review, new developments, and applications. In *Proceedings of the 37th conference on Winter simulation*, Winter Simulation Conference pp 83–95.

[2010] Fua P., Aydin V., Urtasun R., Salzmann M., Least-squares minimization under constraints. Technical Report EPFL-REPORT-150790.

[1980] Fuchs H., Kedem Z.M., Naylor B.F., On visible surface generation by a priori tree structures. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, ACM, New York, pp 124–133.

[1980] Funahashi K., Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, **36**, pp 193–202.

[1989] Funahashi K., On the approximate realization of continuous mappings by neural networks. *Neural Network*, **2**, (3), pp 183–192.

[1993] Funahashi K., Nakamura Y., Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, **6**, pp 801–806.

[2011] Furuya S., Yakubo K., Multifractality of complex networks. *Phys. Rev. E 84*, **84**, (3), 036118.

[2020] Ganesan N., Yu Y., Hientzsch B., Pricing barrier options with DeepBSDEs. Corporate Model Risk, Wells Fargo.

[1994] Gathercole C., Ross P., Dynamic training subset selection for supervised learning in genetic programming. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Manner, editors, *Parallel Problem Solving from Nature III*, volume 866 of LNCS, pp 312–321, Jerusalem. Springer-Verlag.

[1997] Gathercole C., Ross P., Tackling the boolean even N parity problem with genetic programming and limited-error fitness. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, Genetic Programming 1997: Proceedings of the Second Annual Conference, pp 119–127, Stanford University, CA, USA. Morgan Kaufmann.

[2007] Gauci J., Stanley K.O., Generating large-scale neural networks through discovering geometric regularities. In *Proceedings of the 9th annual conference on Genetic and Evolutionary Computation*, pp 1004. ACM.

[1993] Gee A.H., Problem solving with optimization networks. PhD thesis.

[2017] Gehring J., Auli M., Grangier D., Yarats D., Dauphin Y.N., Convolutional sequence to sequence learning. arXiv:1705.03122v2.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1995] Gelman A., Carlin J., Stern H., Rubin D., Bayesian data analysis. Chapman & Hall, 1st edition. 3rd edition in 2013.

[1992] Geman S., Bienenstock E., Doursat R., Neural networks and the bias/variance dilemma. *Neural Computation*, **4**, pp 1–58.

[1997] Gencay R., The predictability of security returns with simple technical trading rules. *Journal of Empirical Finance*, **5**, pp 347–359.

[2018] Gencoglu O., Deep representation learning for clustering of health tweets. arXiv:1901.00439v1.

[1993] George E.I., McCulloch R.E., Variable selection via Gibbs sampling. *J. Amer. Statist. Assoc.*, **88**, pp 881–889.

[1997] George E.I., McCulloch R.E., Approaches for Bayesian variable selection. *Statistica Sinica*, **7**, pp 339–373.

[2000] Gers F.A., Schmidhuber J., Cummins F., Continual prediction with LSTM. *Neural Computation*,**12**, (10), pp 2451–2471.

[2002] Gers F.A., Schraudolph N., Schmidhuber J., Learning precise timing with LSTM recurrent networks. *Journal of Machine Learning Research*, **3**, pp 115–143.

[1998] Gershenfeld N., The nature of mathematical Modeling. Cambridge University Press

[2009] Getoor R., J.L. Doob: Foundations of stochastic processes and probabilistic potential theory. *The Annals of Probability*, **37**, (5), pp 1647–1663.

[2001] Ghahramani Z., An introduction to hidden Markov models and bayesian networks. *International Journal of Pattern Recognition and Artificial Intelligence*, **15**, (1), pp 9–42.

[2004] Ghallab M., Nau D.S., Traverso P., Ghallab, Malik; Nau, Dana S.; Traverso, Paolo (2004), Automated planning: Theory and practice. Morgan Kaufmann Publishers.

[2006] Giggs M.S., Maier H.R., Dandy G.C., Nixon J.B., Minimum number of generations required for convergence of genetic algorithms. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, Vancouver, Canada, pp 2580–2587.

[1959] Gilbert E.N., Random graphs. *Annals of Mathematical Statistics*, **30**, (4), pp 1141–1144.

[1994] Giles L., Omlin C.W., Pruning recurrent neural networks for improved generalization performance. *IEEE transactions on neural networks/a publication of the IEEE Neural Networks Council*, **5**, (5), pp 848–851.

[1996] Gilks W.R., Richardson S., Spiegelhalter D.J., Markov chain Monte Carlo in practice. Chapman & Hall.

[1990] Girosi F., Poggio T., Networks and the best approximation property. *Biological Cybernetics*, **63**, pp 169–176.

[2014] Girschick R., Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, pp 580–587. arXiv:1311.2524.

[2015] Girschick R., Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pp 1440–1448. arXiv:1504.08083.

[2009] Givant S., Halmos P., Introduction to Boolean algebras. Undergraduate Texts in Mathematics, Springer.

[2004] Glasserman P., Monte Carlo methods in financial engineering, Springer-Verlag, New York, Berlin, Heidelberg.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2010] Glorot X., Bengio Y., Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp 249–256.

[2007] Goertzel B., Pennachin C., Artificial general intelligence. Cognitive Technologies, Springer.

[2006] Goh K-I., Salvi G., Kahng B., Kim D., Skeleton and fractal scaling in complex networks. *Phys. Rev. Lett.*, **96**, 018701.

[2018] Golan I., El-Yaniv R., Deep anomaly detection using geometric transformations. In *NeurIPS*, pp 9758–9769.

[1983] Goldberg D.E., Computer-aided pipeline operation using genetic algorithms and rule learning. Doctoral Dissertation, University of Michigan, Ann Arbor, MI.

[1987] Goldberg D.E., Richardson J., Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, J.J., editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pp 148–154. San Francisco. Kaufmann.

[1989] Goldberg D.E., Genetic algorithms in search optimization and machine learning. Addison-Wesley, Reading, MA.

[1989b] Goldberg D.E., Genetic algorithms and Walsh functions: Part 1, A gentle introduction. *Complex Systems*, **3**, pp 129–152.

[1989c] Goldberg D.E., Genetic algorithms and Walsh functions: Part 2, Deception and its analysis. *Complex Systems*, **3**, pp 153–171.

[2016] Goldberg Y., A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, **57**. Also in arXiv:1510.00726 in 2015.

[1997] Gomez F.J., Miikkulainen R., Incremental evolution of complex general behavior. Technical Report AI96-248, The University of Texas at Austin, Department of Computer Sciences.

[1999] Gomez F.J., Miikkulainen R., Solving non-Markovian control tasks with neuroevolution. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pp 1356–1361, San Francisco. Kaufmann.

[2003] Gomez F.J., Robust non-linear control through neuroevolution. PhD thesis.

[2006] Gomez F.J., Schmidhuber J., Miikkulainen R., Efficient non-linear control through neuroevolution. In Johannes Furnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, ECML, volume 4212 of Lecture Notes in Computer Science, pp 654–662. Springer.

[2008] Gomez F.J., Schmidhuber J., Miikkulainen R., Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, **9**, pp 937–965.

[2013] Gomez R.H., Coello Coello C.A., MOMBI: A new metaheuristic for many-objective optimization based on the R2 indicator. In *2013 IEEE Congress on Evolutionary Computation (CEC-2013)*, pp 2488–2495, IEEE Press, Cancun, Mexico, 20-23 June, 2013.

[2017] Gong Y., Luo H., Zhang J., Natural language inference over interaction space. arXiv preprint arXiv:1709.04348.

[2013] Goodfellow I.J., Warde-Farley D., Mirza M., Courville A., Bengio Y., Maxout networks. arXiv:1302.4389.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2014] Goodfellow I.J., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y., Generative adversarial nets. In *Proceedings of the International Conference on Neural Information Processing Systems*, pp 2672–2680.

[2016] Goodfellow I., Bengio Y., Courville A., Deep learning. MIT Press.

[2018] Gorban A.N., Tyukin I.Y., Blessing of dimensionality: Mathematical foundations of the statistical physics of data". *Phil. Trans. R. Soc. A.*, 376, 20170237. arXiv:1801.03421.

[2006] Gosavi A., A risk-sensitive approach to total productive maintenance. *Automatica*, **42**, pp 1321–1330.

[2010] Gosavi A., Finite horizon Markov control with one-step variance penalties. In *Proceedings of the Allerton Conferences*, Allerton, IL.

[2010] Gosavi A., Meyn S.P., Value iteration on two time-scales for variance-penalized Markov control. in *Proceedings of the 2010 IER Conferences*, Cancun, Mexico, A. Johnson and J. Miller, eds.

[2014] Gosavi A., Handbook of simulation optimization. Springer.

[1997] Gottfredson L.S., Mainstream science on intelligence. *Intelligence*, **24**, pp 13–23.

[1998] Gottfredson L.S., The general intelligence factor. *Scientific American Presents*, **9**, (4), pp 24–29.

[2014] Goudarzi A., Banda P., Lakin M.R., Teuscher C., Stefanovic D., A comparative study of reservoir computing for temporal signal processing. Working Paper, University of New Mexico and Portland State University.

[2004] Granger C., Hyng N., Occasional structural breaks and long memory with an application to the SP500 absolute stock returns. *Journal of Empirical Finance*, **11**, pp 399–421.

[1976] Graves J.E., Granville's new strategy of daily stock market timing for maximum profit. Prentice-Hall, Inc.

[2010] Grassl T., A reinforcement learning approach for pricing derivatives. Working Paper

[2007] Grau A.J., Applications of least-square regressions to pricing and hedging of financial derivatives. Ph.D. thesis, Technische Universitat Munchen.

[2001] Gravano L., Ipeirotis P.G., Jagadish H.V., Koudas N., Muthukrishnan S., Srivastava D., Approximate string joins in a database (almost) for free. In *Proc. 27th Int'l Conf. Very Large Databases (VLDB '01)*, pp 491–500.

[2003] Gravano L., Ipeirotis P.G., Koudas N., Srivastava D., Text joins in an RDBMS for Web data integration. In *Proc. 12th Int'l World Wide Web Conf. (WWW12)*, pp 90–101.

[2005] Graves A., Schmidhber J., Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, **18**, (5-6), pp 602–610.

[2005b] Graves A., Fernandez S., Schmidhber J., Bidirectional LSTM networks for improved phoneme classification and recognition. Artificial Neural Networks: Formal Models and Their Applications-ICANN 2005. Springer Berlin Heidelberg, pp 799–804.

[2008] Graves A., Supervised sequence labelling with recurrent neural networks. PhD thesis, Technische Universitat Munchen.

[2008] Graves A., Schmidhber J., Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 21.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2011] Graves A., Mohamed A-R., Hinton G., Speech recognition with deep recurrent neural networks. *International Conference on Acoustics, Speech and Signal Processing*, pp 1033–1040.

[2012] Graves A., Supervised sequence labelling with recurrent neural networks. Studies in Computational Intelligence, Springer-Verlag, Berlin Heidelberg.

[2013] Graves A., Generating sequences with recurrent neural networks. Working Paper, arXiv:1308.0850.

[2014] Graves A., Wayne G., Danihelka I., Neural turing machines. Working Paper, arXiv:1410.5401v2.

[2014b] Graves A., Jaitly N., Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp 1764–1772.

[2004] Grech D., Mazur Z., Can one make any crash prediction in finance using the local Hurst exponent idea? *Physica A: Statistical Mechanics and its Applications*, **336**, (1), pp 133–145.

[2005] Grech D., Mazur Z., Statistical properties of old and new techniques in detrended analysis of time series. *Acta Physica Polonica B*, **36**, (8), pp 2403–2413.

[1995] Green P.J., Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, **82**, (4), pp 711–732.

[1996] Green G.M., Pragmatics and natural language understanding. Routledge, 2 edition.

[2015] Greff K., Srivastava R.K., Koutnik J., Steunebrink B.R., Schmidhuber J., LSTM: A search space odyssey. Transactions On Neural Networks And Learning Systems.

[2014] Gregor K., Danihelka I., Mnih A., Blundell C., Wierstra D., Deep autoregressive networks. In *ICML*, pp 1242–1250.

[2002] Griffiths T.L., Steyvers M., A probabilistic approach to semantic representation. In *Proceedings of the 24th Annual Conference of the Cognitive Science Society*.

[2004] Griffiths T.L., Steyvers M., Finding scientific topics. In *Proceedings of the National Academy of Science*, **101**, pp 5228–5235.

[1992] Grimmett G.R., Stirzaker D.R., Probability and random processes. Oxford Science Publications, Second Edition.

[1976] Grinder J., Bandler R., The structure of magic II. Science and behavior Books Inc.

[2001] Grinder J., Bostic St Clair C., Whispering in the Wind. John Grinder & Carmen Bostic.

[1988] Grotschel M., Lovasz L., Schrijver A., Geometric algorithms and combinatorial optimization. Springer

[1994] Gruau F., Neural network synthesis using cellular encoding and the genetic algorithm. PhD thesis, Laboratoire de l'Informatique du Parallilisme, Ecole Normale Supirieure de Lyon, France.

[1995] Gruau F., Automatic definition of modular neural networks. *Adaptive Behaviour*, **3**, (2), pp 151–183.

[2017] GNU Scientific Library. https://www.gnu.org/software/gsl/.

[2010] Gu G-F., Zhou W-X., Detrending moving average algorithm for multifractals. *Physical Review E*, **82**, 011136, pp 1–12.

[2016] Gu S., Lillicrap T., Sutskever I., Levine S., Continuous deep Q-learning with model-based acceleration: Continuous Q-learning with action-quadratic value functions. Google Brain, Google DeepMind.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2017] Gu S., Lillicrap T., Ghahramani Z., Turner R.E., Levine S., Q-prop: Sample-efficient policy gradient with an off-policy critic. Conference Paper at the International Conference on Learning Representations (ICLR) 2017.

[2014] Gudivada V.N., Rao D., Raghavan V.V., NoSQL systems for big data management. In *2014 IEEE World Congress on Services*, Los Alamitos, CA, USA, IEEE Computer Society, pp 190–197.

[2015] Gudivada V., Baeza-Yates R., Raghavan V., Big data: Promises and problems. *IEEE Computer*, **48**, (3), pp 20–23.

[2015b] Gudivada V.N., Rao D., Raghavan V.V., Big data driven natural language processing research and applications. In *Big Data Analytics*, V. Govindaraju, V. Raghavan, and C. R. Rao, Eds. New York, Elsevier, pp 203–238.

[2017] Gudivada V., Apon A., Ding J., Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations. In *International Journal on Advances in Software*, pp 1–20.

[2019] Gupta H., Srikant R., Ying L., Adaptive learning rate selection for temporal difference learning. In *Real-world Sequential Decision Making Workshop at ICML 2019*.

[2000] Gurevich Y., Sequential abstract state machines capture sequential algorithms, *ACM Transactions on Computational Logic*, **1**, (1), pp 77–111.

[2011] Hafner R., Riedmiller M., Reinforcement learning in feedback control. *Machine Learning*, **84**, (1-2), pp 137–169.

[2002] Hagan P.S., Kumar D., Lesniewski A.S., Woodward D.E., Managing smile risk. *Wilmott Magazine*, pp 84–108.

[2019] Hahn M., Futrell R., Estimating predictive rate-distortion curves via neural variational inference. *Entropy*, **21**, 640, doi:10.3390/e21070640

[2015] Hallac D., Leskovec J., Boyd S., Network lasso: clustering and optimization in large graphs. Stanford University, arXiv:1507.00280.

[2017] Halperin I., QLBS: Q-Learner in the Black-Scholes (-Merton) worlds. NYU Tandon School of Engineerin.

[2018] Halperin I., The QLBS Q-learner goes NuQLear: Fitted Q-Iteration, inverse RL, and option portfolios. NYU Tandon School of Engineerin.

[1986] Halsey T.C., Jensen M.H., Kadanoff L.P., Procaccia I., Sharaiman B.I., Fractal measures and their singularities: The characterization of strange sets. *Phys. Rev. A 33*, **33**, (2), pp 1141.

[2000] Hammer B., On the approximation capability of recurrent neural networks. *Neurocomputing*, **31**, (1-4), pp 107–123.

[1992] Hampo R.J., Marko K.A., Application of genetic programming to control of vehicle systems. In *Proceedings of the Intelligent Vehicles*, 92 Symposium, Detroit, Mi, USA.

[1983] Hampson S.E., A neural model of adaptive behavior. Doctoral dissertation, Department of Information and Computer Science, University of California, Irvine.

[2006] Han J., Kamber M., Data mining: Concepts and techniques. Elsevier ; Morgan Kaufmann, 2nd ed. Amsterdam, Boston, San Francisco, CA.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2013] Han Y., Yang K., Zhou G., A new anomaly: The cross-sectional profitability of technical analysis. *Journal of Financial and Quantitative Analysis*, **48**, (5), pp 1433–1461.

[2017] Han J., Jentzen A., Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning. Working Paper, arXiv:1707.02568.

[2018] Han J., Jentzen A., Weinan E., Solving high-dimensional partial differential equations using deep learning. In *Proceedings of the National Academy of Sciences*, **115**, (34), pp 8505–8510. IEEE Press.

[2015] Han A.L-F., Zeng X., Wong D,F., Chao L,F., Chinese named entity recognition with graph-based semi-supervised learning model. In *Proceedings of SIGHAN workshop in ACL-IJCNLP*.

[1992] Hancock P.J.B., Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification. In *Combinations of Genetic Algorithms and Neural Networks*, COGANN-92. International Workshop, pp 108–122. IEEE.

[2001] Hand D., Mannila H., Smyth P., Principles of data mining. MIT Press.

[1994] Handley S., On the use of a directed acyclic graph to represent a population of computer programs. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, pp 154–159. IEEE Press.

[1982] Hanley J.A., McNeil B.J., The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, **143**, (1), pp 29–36.

[1990] Hansen L.K., Salamon P., Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**, (10), pp 993–1001.

[2016] Harari Y.N., Homo deus: A brief history of tomorrow. Harvill Secker.

[1969] Harary F., Graph theory. Reading, MA: Addison-Wesley.

[1990] Hardle W., Applied nonparametric regression. Cambridge, UK and New York, Cambridge University Press.

[2013] Hargreaves C., Hao Y., Prediction of stock performance using analytical techniques. *Journal of Emerging Technologies in Web Intelligence*, **5**, (2), pp 136–142.

[1997] Harrald P.G., Kamstra M., Evolving artificial neural networks to combine financial forecasts. *IEEE Transactions on Evolutionary Computation*, **1**, (1), pp 40–52.

[2005] Harrington N., The link between bollinger bands and the commodity channel index Harrington. *Technical Analysis of Stocks and Commodities*, **23**, (10), pp 32–38.

[2012] Harris D., Harris S., Digital design and computer architecture. (2nd ed.), San Francisco, Calif., Morgan Kaufmann.

[1979] Harrison J.M., Kreps D., Martingale and arbitrage in multiperiods securities markets. *Journal of Economic Theory*, **20**, pp 381–408.

[1981] Harrison J.M., Pliska S.R., Martingales and stochastic integrals in the theory of continuous trading. *Stochastic Processes Applications*, **11**, pp 215–260.

[2002] Harvey C.R., Travers K.E., Costa M.J., Forecasting emerging market returns using neural networks. *Emerging Markets Quarterly*, Summer, pp 1–12.

[2011] Hassan M.K., Hassan M.Z., Pavel N.I., Dynamic scaling, data-collapse and self-similarity in Barabasi-Albert networks. *Journal of Phys. A: Math. and Theor.*, **44**, (17), 175101.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2009] Hastie T., Tibshirani R., Friedman J., The elements of statistical learning: Data mining, inference, and prediction. Springer Series in Statistics, Second Edition.

[2004] Haupt R.L., Haupt S.E., Practical genetic algorithms. Hoboken, Wiley, 2nd ed.

[1992] Haussler D., Decision theoretic generalizations of the PAC model for neural net and other learning application. *Information and Computation*, **100**, pp 78–150.

[2000] Haykin S., Adaptive filter theory. Prentice-Hall, Upper Saddle River, New Jersey.

[2005] Haykin S., Neural networks: A comprehensive foundation. Pearson Education, New Delhi,

[2008] Hazan E., Kale S., Extracting certainty from uncertainty: Regret bounded by variation in costs. *COLT*. Published in Machine Learning, September 2010, Volume 80, Issue 2-3, pp 165–188.

[2015] He K., Zhang X., Ren S., Sun J., Delving deep into recti

ers: Surpassing human, level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pp 1026–1034.

[2015b] He H., Gimpel K., Lin J., Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp 1576–1586.

[2016] He K., Zhang X., Ren S., Sun J., Deep residual learning for image recognition. In *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pp. 770–778.

[1949] Hebb D.O., The organization of behavior. Wiley, New York.

[1969] Heesch H., Untersuchungen zum Vierfarbenproblem. Mannheim: Bibliographisches Institut.

[1998] Helstrom T., Holmstrom K., Predicting the stock market. Published as Opuscula ISRN HEV-BIB-OP-26-SE.

[1983] Hentschel H.G.E., Procaccia I., The infinite number of generalised dimensions of fractals and strange attractors. *Physica 8D*, pp 435–444.

[2017] Hernandez A., Model calibration with neural networks. Risk.

[2008] Herschel M., Naumann F., Scaling up duplicate detection in graph data. In *CIKM'08*, October 26-30, Napa Valley, California, USA.

[1952] Hestenes M.R., Stiefel E., Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, **49**, pp 409–436.

[1993] Heston S., A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Review of Financial Studies*, **6**, pp 327–343.

[2019] Hientzsch B., Introduction to solving quant finance problems with time-stepped FBSDE and deep learning. Working Paper, arXiv:1911.12231.

[2000] Hill J.R., Pruitt G., Hill L., The ultimate trading guide. John Wiley & Sons, Wiley Trading Advantage.

[2005] Hinrichsen D., Pritchard A.J., Mathematical systems theory I: Modelling, state space analysis, stability and robustness. Springer.

[1994] Hinton G.E., Zemel R.S., Autoencoders, minimum description length and Helmholtz free energy. In *Advances in neural information processing systems*, **6**, pp 3–10.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1995] Hinton G.E., Dayan P., Frey B.J., Neal R.M., The wake-sleep algorithm for unsupervised neural networks. *Science*, **268**, pp 1558–1161.

[2006] Hinton G.E., Salakhutdinov R.R., Reducing the dimensionality of data with neural networks. *Science*, **313**, (5786) pp 504–507.

[2006b] Hinton G.E., Osindero S., Teh Y-W., A fast learning algorithm for deep belief nets. To appear in Neural Computation 2006.

[2012] Hinton G.E., Srivastava N., Krizhevsky A., Sutskever I., Salakhutdinov R.R., Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580.

[1994] Ho T.K., Hull J.J., Srihari S.N., Decision combination in multiple classifier systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**, (1), pp 66–75.

[1995] Ho T.K., Random decision forests. in *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, Montreal, pp 14–16

[1998] Ho T.K., The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**, (8), pp 832–844

[2004] Ho D-S., Lee C-K., Wang C-C., Chuang M., Scaling characteristics in the Taiwan stock market. *Physica A*, **332**, pp 448–460.

[2003] Hoai N.X., McKay R.I., Abbass H.A., Tree adjoining grammars, language bias, and genetic programming. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, Genetic Programming, *Proceedings of EuroGP'2003*, volume 2610 of LNCS, pp 335–344, Springer-Verlag.

[1997] Hochreiter S., Schmidhuber J., Long short-term memory. *Neural Computation*, **9**, (8), pp 1735–1780.

[2001] Hochreiter S., Bengio Y., Frasconi P., Schmidhuber J., Gradient flow in recurrent nets: The difficulty of learning long-term dependencies. In *A field guide to dynamical recurrent neural networks*, IEEE Press.

[1997] Hodges S.D., A generalisation of the Sharpe ratio and its applications to valuation bounds and risk measures. Financial Options Research Centre, Working Paper, University of Warwick.

[2004] Hodge V.J., Austin J., A survey of outlier detection methodologies. *Artificial Intelligence Review*, **22**, (2), pp 85–126.

[1999] Hofmann T., Probabilistic latent semantic indexing. In *Proceedings of the Twenty-Second Annual International SIGIR Conference*.

[2001] Hofmann T., Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning Journal*, **42**, (1), pp 177–196.

[1962] Holland J.H., Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, **9**, pp 297-314.

[1973] Holland J.H., Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computation*, **2**, pp 88–105.

[1975] Holland J.H., Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor.

[1986] Holland J.H., Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.), Machine learning: An artificial intelligence approach (Vol 2). Los Altos, CA, Morgan Kaufmann.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1982] Hopfield J.J., Neural networks and physical systems with emergent collective computational abilities. *Proceedings of National Academy of Sciences USA*, **79**, pp 2554–2558.

[1985] Hopfield J.J., Tank D.W., Neural computation of decisions in optimization problems. *Biological Cybernetics*, **52**, pp 141–152.

[1987] Hopfield J.J., Tank D.W., Computing with neural circuits: A model. *Science*, **233**, pp 625–633.

[1983] Hormander L., The analysis of linear partial differential operators I. Grundl. Math. Wissenschaft., 256, Springer.

[1997] Horn J., The nature of niching: Genetic algorithms and the evolution of optimal, cooperative populations. PhD thesis, University of Illinois.

[1989] Hornik K., Stinchcombe M., White H., Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**, (5), pp 359–366.

[1990] Hornik K., Stinchcombe M., White H., Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, **3**, (5), pp 551–560.

[1991] Hornik K., Approximation capabilities of multilayer feedforward networks. *Neural Networks*, **4**, (2), pp 251–257.

[2005] Horton J.C., Adams D.L., The cortical column: A structure without a function. *Philosophical Transactions of the Royal Society B: Biological Sciences*, **360**, (1456), pp 837.

[2019] Horvath B., Muguruza A., Tomas M., Deep learning volatility: A deep neural network perspective on pricing and calibration in (rough) volatility models. Preprint, arXiv:1901.09647v1.

[2019] Houlsby N., Giurgiu A., Jastrzebski S., Morrone B., de Laroussilhe Q., Gesmundo A., Attariyan M., Gelly S., Parameter-efficient transfer learning for NLP. Working Paper, arXiv:1902.00751

[1960] Howard R., Dynamic programming and Markov processes. MIT Press, Cambridge, MA.

[1971] Howard R.A., Dynamic probabilistic systems, Vol II: Semi-Markov and decision processes. Wiley, New York.

[2018] Howard J., Ruder S., Universal language model fine-tuning for text classification. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp 328–339.

[2008] Hsieh C-J., Chang K-W., Lin C-J., Keerthi S.S., Sundararajan S., A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the 25th International Conference on Machine Learning (ICML) 08*, New York, NY, USA, pp 408–415.

[1988] Huang C., Litzenberger R.H., Foundations for financial economics. North-Holland.

[2017] Huang X., Li Y., Poursaeed O., Hopcroft J.E., Belongie S.J., Stacked generative adversarial networks. In *CVPR*, **2**, pp 3.

[2017b] Huang G., Liu Z., Van Der Maaten L., Weinberger K.Q., Densely connected convolutional networks. In *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pp. 2261–2269.

[1968] Hubel D.H., Wiesel T.N., Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, **195**, (1), pp 215–243.

[2005] Hubel D.H., Wiesel T.N., Brain and visual perception: The story of a 25-year collaboration. Oxford University Press US, pp 106.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2007] Hubner G., How do performance measures perform? EDHEC Business School, March.

[2017] Hudson R., Urquhart Y., Wang P., Why do moving average rules work? Comprehensive evidence from world markets.

[1987] Hull J.C., White A., The pricing of options on assets with stochastic volatilities. *Journal of Finance*, **42**, (2), pp 281–300.

[1990] Hull J., White A., Pricing interest rate derivative securities. *The Review of Financial Studies*, Volume 4, **3**, pp 573–592. Reprinted in Vasicek and Beyond, pp 129–144. London: Risk Publications 1996.

[2012] Hull J.C., Options, futures and other derivatives. Prentice & Hall, Pearson Education, Eight Edition.

[2004] Hulse M., Wischmann S., Pasemann F., Structure and function of evolved neuro-controllers for autonomous robots. *Connection Science*, **16**, (4), pp 249–266.

[2019] Humeau S., Shuster K., Lachaux M-A., Weston J., Real-time inference in multi-sentence tasks with deep pretrained transformers. arXiv preprint arXiv:1905.01969, abs/1905.01969.

[1951] Hurst H.E., Long-term storage capacity of reservoirs. *Trans. Amer. Soc. Civil Engineers*, **116**, pp 770–799.

[1965] Hurst H.E., Black R.P., Simaika Y.M., Long-term storage: An experimental study. London, Constable.

[1994] Hutchinson J.M., Lo A.W., Poggio T., A nonparametric approach to pricing and hedging derivative securities via learning networks. *Journal of Finance*, **49**, (3), pp 851–889.

[2006] Hyndman R.J., Koehler A.B., Another look at measures of forecast accuracy. *International Journal of Forecasting*, **22**, pp 679–688.

[2012] Ihlen E.A.F., Introduction to multifractal detrended fluctuation analysis in Matlab. *Frontiers in Physiology*, **3** (141), pp 1–18.

[2013] Ihlen E.A.F., Multifractal analyses of response time series: A comparative study. *Behav. Res.*, **45**, pp 928–945.

[2014] Ihlen E.A.F., Vereijken B., Detection of co-regulation of local structure and magnitude of stride time variability using a new local detrended fluctuation analysis. *Gait & Posture*, **39**, pp 466–471.

[2000] Ilkka T., Data is more than knowledge. *Journal of Management Information Systems*, **6**, (3), pp 103–117.

[2003] Ilonen J., Kamarainen J.K., Lampinen J., Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters*, **17**, pp 93–105.

[2007] Inden B., Stepwise transition from direct encoding to artificial ontogeny in neuroevolution. In Fernando Almeida e Costa, Luis Mateus Rocha, Ernesto Costa, Inman Harvey, and Antonio Coutinho, editors, ECAL, volume 4648 of Lecture Notes in Computer Science, pp 1182–1191. Springer.

[2008] Inden B., Neuroevolution and complexifying genetic architectures for memory and control tasks. *Theory in Biosciences*, **127**, (2), pp 187–194.

[2010] Inden B., Jin Y., Haschke R., Ritter H., Neatfields: Evolution of neural fields. In *Genetic and Evolutionary Computation Conference*, pp 645–646.

[2016] Inthachot M., Boonjing V., Intakosum1 S., Artificial neural network and genetic algorithm hybrid intelligence for predicting Thai ttock price index trend. *Computational Intelligence and Neuroscience*, **2016**, pp 1–8.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2015] Ioffe S., Szegedy C., Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

[2003] Irvine A.D., Principia mathematica (Stanford Encyclopedia of Philosophy). Metaphysics Research Lab, CSLI, Stanford University.

[2011] Isele R., Bizer C., Learning linkage rules using genetic programming. In *6th International Workshop on Ontology Matching*, Bonn,Germany, pp 13–24.

[2004] Ishii K., van der Zant T., Becanovic V., Ploger P., Identification of motion with echo state network. in *Proceedings of the OCEANS 2004 MTS/IEEE Conference*, **3**, pp 1205–1210.

[1967] Ivakhnenko A.G., Lapa V.G., Cybernetics and forecasting techniques. American Elsevier Pub. Co.

[1971] Ivakhnenko A.G., Polynomial theory of complex systems. IEEE Transactions on Systems, Man and Cybernetics, **1**, (4), pp 364–378.

[1962] Ivanov V.K., On linear problems which are not well-posed. *Soviet Math. Dokl*, **3**, pp 981–983.

[1999] Jackwerth J.C., Option-implied risk-neutral distributions and implied binomial trees: A literature review. *Journal of Derivatives*, **7**, (2), pp 66–81.

[1991] Jacobs R.A., Jordan M.I., Nowlan S.J., Hinton G.E., Adaptive mixtures of local experts. *Neural Computation*, **3**, (1), pp 79–87.

[1987] Jacod J., Shiryaev A.N., Limit theorems for stochastic processes. Springer.

[1998] Jaeger H., Discrete-time, discrete-valued observable operator models: A tutorial. Technical Report, German National Research Center for Information Technology.

[2001] Jaeger H., The echo state approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology.

[2002] Jaeger H., Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach. GMD-Forschungszentrum Informationstechnik, volume 5.

[2003] Jaeger H., Adaptive nonlinear system identification with Echo State Networks. International University Bremen.

[2004] Jaeger H., Seminar slides. Seminar Spring.

[2004] Jaeger H., Haas H., Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, **304**, pp 78–80.

[2007] Jaeger H., Echo state network. *Scholarpedia*, **2**, (9), pp 2330.

[2012] Jaeger H., Long short-term memory in echo state networks: Details of a simulation study. Technical report, Jacobs University Bremen.

[2001] Jain B.J., Pohlheim H., Wegener J., On termination criteria of evolutionary algorithms. GECCO 2001 - Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kauffmann, San Francisco.

[2016] Jamous R.A., El. Seidy E., Bayoum B.I., A novel efficient forecasting of stock market using particle swarm optimization with center of mass based technique. *International Journal of Advanced Computer Science and Applications*, **7**, (4), pp 342–347.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1991] Jamshidian F., Bond and option evaluation in the Gaussian interest rate model. *Research in Finance*, **9**, pp 131–170. Reprinted in Vasicek and Beyond, pp 161–190. London: Risk Publications 1996.

[2017] Janocha K., Czarnecki W.M., On loss functions for deep neural networks in classification. Working Paper, Jagiellonian University, Krakow, Poland.

[1991] Jansen D.W., de Vries C.G., On the frequency of large stock market returns: Putting booms and busts into perspective. *Review of Economics and Statistics*, **73**, (1), pp 18–24.

[2016] Janssen S.C., Reducing labeled data usage in duplicate detection using deep belief networks. Master Thesis, Mathematics and Computer Science, University of Twente.

[1970] Jaro M.A., Unimatch: A record linkage system - User's manual. Technical Report, US Bureau of the Census, Washington, D.C.,.

[1989] Jaro M.A., Advances in record-linkage methodology as applied to matching the 1985 Census of Tampa, Florida. *J. Am. Statistical Assoc.*, **84**, (406), pp 414–420.

[2009] Jarrett K., Kavukcuogl K., Ranzato M., LeCun Y., What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pp 2146–2153.

[2004] Jebara T., Machine learning: Discriminative and generative. The Springer International Series in Engineering and Computer Science.

[1969] Jelinkek F., A fast sequential decoding algorithm using a stack. *IBM J. Res. Develop.*, **13**, pp 675–685.

[1968] Jensen M.J., The performance of mutual funds in the period 1945-1964. *Journal of Finance*, **23**, No 2, pp 389–416.

[2001] Jeong H., Neda Z., Barabasi A-L., Measuring preferential attachment for evolving networks. *Europhys. Lett.*, **61**, pp 567–572.

[1874] Jevons W.S., The principles of science: A treatise on logic and scientific method. Macmillan & Co., London, 1874, 2nd ed. 1877, 3rd ed. 1879. Reprinted with a foreword by Ernst Nagel, Dover Publications, New York, 1958.

[2011] Jhankal N.K., Adhyaru D., Bacterial foraging optimization algorithm: A derivative free technique. Engineering (NUiCONE), Nirma University International Conference on 2011.

[2008] Jiang F., Berry H., Schoenauer M., Supervised and evolutionary learning of echo state networks. In *Proceedings of 10th International Conference on Parallel Problem Solving from Nature*, **5199** of LNCS, pp 215–224, Springer.

[2012] Jizba P., Korbel J., Methods and techniques for multifractal spectrum estimation in financial time series. FNSPE, Czech Technical University, Prague.

[2009] Johnson M., How the statistical revolution changes (computational) linguistics. In *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics*.

[2006] Jong N., Stone P., Kernel-based models for reinforcement learning in continuous state spaces. In *ICML workshop on Kernel Machines and Reinforcement Learning*, June.

[2004] Jordaan E., Kordon A., Chiang L., Smits G., Robust inferential sensors based on ensemble of predictors generated by genetic programming. In Xin Yao, Edmund Burke, Jose A. Lozano, Jim Smith, Juan J. Merelo-Guervos, John A. Bullinaria, Jonathan Rowe, Peter Tino Ata Kaban, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of LNCS, pp 522–531, Birmingham. Springer-Verlag.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1992] Jordan M.I., Jacobs R.A., Hierarchies of adaptive experts. In J.E. Moody, S.J. Hanson, and R. Lippmann, editors, Advances in *Neural Information Processing Systems 4*, Morgan Kaufmann, San Francisco, pp 985–992.

[1995] Jordan M.I., Xu L., Convergence results for the EM approach to mixtures of experts architectures. *Neural Networks*, **8**, (9), pp 1409–1431.

[1999] Jordan M.I., Learning in graphical models. Edited by M.I. Jordan, MIT Press.

[1999] Jordan M.I., Ghahramani Z., Jaakkola T., Saul L.K., An introduction to variational methods for graphical models. *Machine Learning*, **37**, (9), pp 183–233.

[2013] Jordehi A., Jasni J., Parameter selection in particle swarm optimisation: A survey. *Journal of Experimental & Theoretical Artificial Intelligence*, **25**, (4), pp 527–542.

[2016] Jozefowicz R., Vinyals O., Schuster M., Shazeer N., Wu Y., Exploring the Limits of Language Modeling. Cornell University.

[2007] Junyou B., Stock price forecasting using PSO-trained neural networks. In *IEEE Congress on Evolutionary Computation*, pp 2879–2885.

[2009] Jurafsky D., Martin J.H., Speech and language processing. Pearson Education International, 2nd edition.

[1999] Kaboudan M., A measure of time series predictability using genetic programming applied to stock returns. *Journal of Forecasting*, **18**, pp 345–357.

[2000] Kaboudan M., Genetic programming prediction of stock prices. *Computational Economics*, **6**, (3), pp 207–236.

[2005] Kaboudan M., Extended daily exchange rates forecasts using wavelet temporal resolutions. *New Mathematics and Natural Computing*, **1**, pp 79–107.

[1996] Kaelbling L.P., Littman M.L., Moore A.W., Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, **4**, pp 237–285.

[1962] Kahn A.B., Topological sorting of large networks. *Communications of the ACM*, **5**, (11), pp 558–562.

[2015] Kaiming H., Zhang X., Ren S., Sun J., Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. arXiv:1502.01852.

[1997] Kainen P.C., Utilizing geometric anomalies of high dimension: When complexity makes computation easier. in Karny, M., Warwick, K. (eds.), *Computer Intensive Methods in Control and Signal Processing*, pp 283–294.

[2002] Kakade S., Langford J., Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pp 267–274.

[2005] Kalai A., Servedio R., Boosting in the presence of noise. *Journal of Computer and System Sciences*, **71**, (3), pp 266–290.

[2013] Kalchbrenner N., Blunsom P., Recurrent continuous translation models. In *EMNLP*, 2013.

[2015] Kalchbrenner N., Danihelka I., Graves A., Grid long Short-Term memory. arXiv:1507.01526.

[2015] Kaleel S.B., Abhari A., Cluster-discovery of twitter messages for event detection and trending. *Journal of Computational Science*, **6**, pp 47–57.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1960] Kalman R.E., A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, **82**, (1), pp 35–45.

[1986] Kalton G., The treatment of missing survey data. *Survey Methodology*, **12**, pp 1–16.

[2011] Kamber H., Jaiwei P., Jian M., Data mining: Concepts and techniques. Morgan Kaufmann, (3rd ed.).

[2005] Kamphaus R.W., Clinical assessment of child and adolescent intelligence. Springer.

[2002] Kantelhardt J., Zschiegner S., Koscielny-Bunde E., Bunde A., Havlin S., Stanley H.E., Multifractal detrended fluctuation analysis of nonstationary time series. *Physica A*, **316**, (1-4), pp 87–114.

[2011] Kara Y., Boyacioglu M.A., Baykan O.K., Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange. *Expert Systems with Applications*, **38**, (5), pp 5311–5319.

[2004] Karaboga D., Okdem S., A simple and global optimization algorithm for engineering problems: Differential evolution algorithm. *Turkish Journal of Electrical Engineering & Computer Sciences*, **12**, (1), pp 53–60.

[2019] Karatas T., Oskoui A., Hirsa A., Supervised deep neural networks for pricing/calibration of vanilla/exotic options under various different processes. Working Paper.

[2012] Karimi H., Yousefi F., Application of artificial neural network-genetic algorithm (ANN-GA) to correlation of density in nanofluids. *Fluid Phase Equilibria*, **336**, pp 79–83.

[2001] Karlsson L., Conditional progressive planning under uncertainty. *IJCAI*, pp 431–438.

[1996] Keijzer M., Efficiently representing populations in genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming*, 2, chapter 13, pp 259–278. MIT Press, Cambridge, MA, USA.

[1956] Kelly J.L., A new interpretation of information rate. *Bell System Technical Journal*, **35**, pp 917–926.

[1960] Keltner C.W., How to make money in commodities. Keltner Statistical Service.

[1988] Kennedy M.P., Chua L.O., Neural networks for non-linear programming. *IEEE Trans. Circuit and Systems*, **35**, pp 554–562.

[1995] Kennedy J., Eberhart R.C., Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, Australia, pp 1942–1948.

[2001] Kennedy J., Eberhart R.C., Shi Y., Swarm intelligence. Morgan Kaufmann / Academic Press.

[1983] Kent W., A simple guide to five normal forms in relational database theory. *Communications of the ACM*, **26**, pp 120–125.

[2003] Keskustalo H., et al., Non-adjacent digrams improve matching of cross-lingual spelling variants. In *Proceedings of the International Symposium on String Processing and Information Retrieval*, Springer.

[2019] Khandelwal R., K fold and other cross-validation techniques. Retrieved from https://medium.com/datadriveninvestor/k-fold-and-other-cross-validation techniques-6c03a2563f1e.

[1952] Kiefer J., Wolfowitz J., Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, **23**, (3), pp 462–466.

[2000] Kim K-J., Han I., Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications*, **19**, (2), pp 125–132.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2004] Kim K., Choi J-S., Yoon S-M., Multifractal measures for the yen-dollar exchange rate. *Journal of the Korean Physical Society*, **44**, (3), pp 643–646.

[2004b] Kim K-J., Won Boo L., Stock market prediction using artificial neural networks with optimal feature transformation. *Neural computing and applications*, **13**, (3), pp 255–260.

[2006] Kim K-J., Artificial neural networks with evolutionary instance selection for financial forecasting. *Expert Systems with Applications*, **30**, (3), pp 519–526.

[2006] Kim M-J., Min S-H., Han I., An evolutionary approach to the combination of multiple classifiers to predict a stock price index. *Expert Systems with Applications*, **31**, (2), pp 241–247.

[2007] Kim J.S., Goh K-I., Salvi G., Oh E., Kahng B., Kim D., Fractality in complex networks: Critical and supercritical skeletons. *Phys. Rev. E*, **75**, 016110.

[2014] Kim Y., Convolutional neural networks for sentence classification. In *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*.

[2017] Kim Y., Denton C., Hoang L., Rush A.M., Structured attention networks. In *International Conference on Learning Representations*.

[1990] Kimoto T., Asakawa K., Yoda M., Takeoka M., Stock market prediction system with modular neural netorks. In *Proceedings of the 1990 International Joint Conference on Neural Networks*, **1**, Washington, DC, USA, pp 1–6.

[2014] Kingma D.P., Welling M., Auto-encoding variational bayes. In *Proceedings of International Conference on Learning Representations*, (ICLR).

[2014b] Kingma D.P., Welling M., Stochastic gradient VB and the variational auto-encoder. In *Proceedings of the International Conference on Learning Representations*.

[2015] Kingma D.P., Ba J.L., Adam: A method for stochastic optimization. In *Proceedings of ICLR*.

[2019] Kingma D.P., Welling M., An introduction to variational autoencoders. Foundations and Trends in Machine Learning.

[1994] Kinnear K.E., Alternatives in automatic function definition: A comparison of performance. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 6. MIT Press.

[1994b] Kinnear K.E., Fitness landscapes and difficulty in genetic programming. In *Proceedings of the 1994 IEEE World Conference on Computational Intelligence*, volume 1, pp 142–147. IEEE Press.

[1970] Kirk D.E., Optimal control theory: An introduction. Englewood Cliffs, Prentice-Hall.

[1983] Kirkpatrick S., Gelatt C.D., Vecchi M.P., Optimization by simulated annealing. *Science*, **220**, pp 671–680.

[2017] Kirkpatrick J., Pascanu R., Rabinowitz N., Veness J., Desjardins G., etc. Overcoming catastrophic forgetting in neural networks. *PNAS*.

[2015] Kiros R., Zhu Y., Salakhutdinov R.R., Zemel R., Urtasun R., Torralba A., Fidler S., Skip-thought vectors. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, 28, pp 3294–3302, Curran Associates, Inc.

[1998] Kittler J., Hatef M., Duin R., Matas J., On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**, (3), pp 226–239.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1997] Kivinen J., Warmuth M., Exponential gradient versus gradient descent for linear predictors. *Journal of Information and Computation*, **132**, (1), pp 1–63.

[1943] Kleene S.C., Recursive predicates and quantifiers. *American Mathematical Society Transactions*, **54**, (1), pp 41–73.

[1956] Kleene S.C., Representation of events in nerve nets and finite automata. *Annals of Mathematics Studies*, **34**, Princeton University Press, pp 3–41.

[2013] Klenke A., Probability theory: A comprehensive course. Universitext, Second edition, Springer.

[1972] Klopf A.H., Brain function and adaptive systems: A heterostatic theory. Air Force Cambridge Research Laboratories Research Report, AFCRL-72-0164, Bedford, MA.

[1982] Klopf A.H., The hedonistic neuron: A theory of memory, learning, and intelligence. Washington, DC, Hemisphere.

[1973] Knuth D.E., The art of computer programming. Second Edition, Volume 1/Fundamental Algorithms (2nd ed.). Addison-Wesley Publishing Company.

[1998] Kodjabachian J., Meyer J-A., Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects. *Neural Networks*, IEEE Transactions on, **9**, (5), pp 796–812.

[1996] Kohavi R., Wolpert D.H., Bias plus variance decomposition for zero one loss functions. In *Proceedings of the 13th International Conference on Machine Learning*, Bari, Italy, pp 275–283.

[1998] Kohavi R., Provost F., Glossary of terms. *Machine Learning*, **30**, pp 271–274.

[1990] Kohonen T., The self-organizing map. In *Proceedings of the IEEE*, **78**, (9), pp 1464–1480.

[2009] Koller D., Friedman N., Probabilistic graphical models. Massachusetts, MIT Press.

[1931] Kolmogorov A.N. Uber die analytischen Methoden in der Wahrscheinlichkeitstheorie, [On Analytical Methods in the Theory of Probability]. *Mathematische Annalen* (in German). **104**, (1): pp 415–458 [pp 448–451].

[1933] Kolmogorov A.N., Grundbegriffe der Wahrscheinlichkeitsrechnung. Berlin, Germany: Springer. [Transl. Foundations of the theory of probability by N. Morrison, 2nd edn. New York, NY: Chelsea, 1956].

[2000] Konda V.R., Tsitsiklis J.N., Actor-critic algorithms. In *Proceedings of NIPS 12*, pp 1008–1014.

[2008] Konishi S., Kitagawa G., Information criteria and statistical modeling. Springer.

[1989] Koppelberg S., General Theory of Boolean Algebras. Handbook of Boolean Algebras, Vol. 1 (ed. J. Donald Monk with Robert Bonnet). Amsterdam, North Holland.

[1989] Koza J.R., Hierarchical genetic algorithms operating on populations of computer programs. editor N.S. Sridharan, in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI*, **1**, pp 768–774.

[1990] Koza J.R., Non-linear genetic algorithms for solving problems. United States Patent 4935877.

[1992] Koza J.R., Genetic programming: On the programming of computers by means of natural selection. MIT Press, Cambridge, MA, USA.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1994] Koza J.R., Genetic programming II: Automatic discovery of reusable programs. MIT Press, Cambridge Massachusetts.

[1995] Koza J.R., Andre D., Parallel genetic programming on a network of transputers. Technical Report CS-TR-95-1542, Stanford University, Department of Computer Science, January 1995.

[1999] Koza J.R., Andre D., Bennett F.H., Keane M., Genetic programming 3: Darwinian invention and problem solving. Morgan Kaufman.

[1991] Kramer M.A. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, **37**, (2), pp 233–243.

[2000] Krapivsky P.L., Redner S., Leyvraz F., Connectivity of growing random networks. *Physical Review Letter*, **85**, (21), 4629.

[2001] Krapivsky P.L., Redner S., Organization of growing random networks. *Physical Review E*, **63**, (6), 066123.

[2010] Krioukov D., Papadopoulos F., Kitsak M., Vahdat A., Boguna M., Hyperbolic geometry of complex networks. *Physical Review E*, **82**, (3), 036106.

[1971] Kripke S., Identity and necessity. In M.K. Munitz, ed., New York, New York University Press, pp 135–64.

[2013] Kristoufek L., Vosvrda M., Measuring capital market efficiency: Global and local correlations structure. *Physica A*, **392**, (1), pp 184–193.

[2017] Krizhevsky A., Sutskever H., Hinton G.E., ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, **60**, (6), pp 84–90. First published in *NeurIPS* in 2012.

[2007] Kroenke D.M., Auer D.J., Database concepts. 3rd ed. New York, Prentice.

[2010] Krollner B., Vanstone B., Finnie G., Financial time series forecasting with machine learning techniques: A survey. in *ESANN 2010 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, Bruges, Belgium.

[2006] Kucukemre A.U., Echo state networks for adaptive filtering. Master Thesis, University of Applied Sciences, Bohn-Rhein-Sieg, Germany.

[1951] Kuhn H.W., Tucker A.W., Nonlinear programming. In *Proceedings of 2nd Berkeley Symposium*, Berkeley, University of California Press, pp 481–492.

[2018] Kuhn M., Johnson K., Applied predictive modeling. Springer, 1st ed. 2013, Corr. 2nd printing 2018 Edition.

[1992] Kukich K., Techniques for automatically correcting words in text. *ACM Computing Surveys*, **24**, (4), pp 377–439.

[1999] Kumar R., Raghavan P., Rajalopagan S., Tompkins A., The web as a graph. In *Proceedings of the 9th ACM Symposium on Principles of Database Systems*, pp 1–10.

[2000] Kumar R., Raghavan P., Stochastic models for the web graph. *Foundations of Computer Science*, 41st Annual Symposium, pp 57–65.

[2003] Kuncheva L.I., Whitaker C.J., Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, **51**, (2), pp 181–207.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2015] Kundu A., Paul A.K., Shill P.C., Murase K., A multi-objective genetic algorithm with fuzzy relational clustering for automatic data clustering. In *2015 2nd International Conference on Electrical Information and Communication Technologies (EICT)*.

[2016] Kurdi M.Z., Natural language processing and computational linguistics: Speech, morphology, and syntax. Volume 1. ISTE-Wiley.

[2017] Kurdi M.Z., Natural language processing and computational linguistics: Semantics, discourse, and applications. Volume 2. ISTE-Wiley.

[1992] Kurkova V., Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, **5**, pp 501–506.

[2012] Kusakci A.O., Can M., Constrained optimization with evolutionary algorithms: A comprehensive review. *Southeast Europe Journal of Soft Computing*, **1**, (2), pp 16–24.

[2015] Kusner M., Sun Y., Kolkin N., Weinberger K., From word embeddings to document distances. In *International Conference on Machine Learning*, pp 957–966.

[1997] Lagoudakis M.G., Neural networks and optimization problems A case study: The minimum cost spare allocation problem. The Center for Advanced Computer Studies, University of Southwestern Louisiana.

[2017] Lai G., Xie Q., Liu H., Yang Y., Hovy E., Race: Large-scale reading comprehension dataset from examinations. In *EMNLP*.

[2018] Lakshminarayanan C., Szepesvari C., Linear stochastic approximation: How far does constant step-size and iterate averaging go? In *International Conference on Artificial Intelligence and Statistics*, pp 1347–1355.

[1997] Lam L., Suen S.Y., Application of majority voting to pattern recognition: An analysis of its behavior and performance. *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans*, **27**, (5), pp 553–568.

[2014] Lam H.K., Ekong U., Xiao B., Ouyang G., Liu H., Chan K.Y., Ling S.H., Variable weight neural networks and their applications on material surface and epilepsy seizure phase classifications. Working Paper, King's College, London.

[1980] Lambert D.R., Commodity channel index: Tool for trading cyclic trends. Commodities Magazine, 219 Parkade, Cedar Falls, IA 50613.

[2004] Lampinen J., Storn R., Differential evolution. In *New Optimization Techniques in Engineering*, G.C. Onwubolu and B. Babu, Eds., Springer-Verlag, Berlin, pp 123–166.

[2019] Lample G., Conneau A., Cross-lingual language model pretraining. In *CoRR*, abs/1901.07291. In Proceedings of 2019 NeurIPS.

[2018] Lan W., Xu W., Character-based neural networks for sentence pair modelling. In *Proceedings of NAACL-HLT 2018*, pp 157–163, New Orleans, Louisiana.

[2006] Landa Becerra R., Coello Coello C.A., Cultured differential evolution for constrained optimization. *Computer Methods in Applied Mechanics and Engineering*, **195**, July, pp 4303–4322.

[1961] Landauer R., Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, **5**, (3), pp 183–191.

[1998] Landauer T.K., Foltz P.W., Laham D., Introduction to latent semantic analysis. *Discourse Processes*, **25**, pp 259–284.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1984] Lane G., Lane's stochastics. *Technical Analysis of Stocks and Commodities*, pp 87–90.

[1990] Lang K.J., Waibel A.H., Hinton G.E., A time-delay neural network architecture for isolated word recognition. *Neural Networks*, **3**, pp 33–43.

[1995] Langdon W.B., Evolving data structures using genetic programming. Research Note RN/1/95, UCL, January 1995. Accepted for presentation at ICGA-95.

[1995] Langdon W.B., Qureshi A., Genetic programming: Computers using natural selection to generate programs. University College London.

[1998] Langdon W.B., Genetic programming and data structures: Genetic programming + data structures = automatic programming. Volume 1 of Genetic Programming, Kluwer, Boston.

[2000] Langdon W.B., Size fair and homologous tree genetic programming crossovers. *Genetic Programming and Evolvable Machines*, **1**, (1/2), pp 95–119.

[2002] Langdon W.B., Poli R., Foundations of genetic programming. Springer-Verlag.

[2005] Langdon W.B., Banzhaf W., Repeated sequences in linear genetic programming genomes. *Complex Systems*, **15**, (4), pp 285–306.

[1987] Lapedes A., Farber R., Nonlinear signal processing using neural networks: Prediction and modeling. Technical Report, LA-UR87-2662, Los Alamos, New Mexico.

[2010] Larochelle H., Hinton G.E., Learning to combine foveal glimpses with a third-order Boltzmann machine. In *24th Annual Conference on Neural Information Processing Systems 2010*.

[2001] Larsen M.D., Rubin D.B., Alternative automated record linkage using mixture models. *Journal of the American Statistical Association*, **79**, pp 32–41.

[2009] Larson R., Edwards B.H., Calculus. 9th ed., Brooks/Cole.

[1998] Lawler J., Dry H.A., Using computers in linguistics: A Practical Guide Paperback. Routledge, 1 edition.

[2014] Le Q., Mikolov T., Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning*, Beijing, China, 2014. JMLR: W&CP volume 32.

[2016] Lea C., Vidal R., Reiter A., Hager G.D., Temporal convolutional networks: A unified approach to action segmentation. European Conference on Computer Vision. Springer, Cham. arXiv:1608.08242.

[1985] LeCun Y., A learning scheme for asymmetric threshold networks. In *Proceedings of Cognitiva 85*, pp 599–604, Paris, France.

[1986] LeCun Y., Learning processes in an asymmetric threshold network. In F. Fogelman-Soulie, E. Bienenstock, and G. Weisbuch, editors, *Disordered systems and biological organization*, pp 233–240, Les Houches, France, Springer-Verlag.

[1988] LeCun Y., A theoretical framework for back-propagation. In D. Iouretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pp 21–28, CMU, Pittsburgh, Pa, Morgan Kaufmann.

[1989] LeCun Y., Boser B., Denker J.S., Henderson D., Howard R.E., Hubbard W., Jackel L.D., Backpropagation applied to handwritten zip code recognition. *Neural Computation*, **1**, (4), pp 541–551.

[1998] LeCun Y., Bottou L., Bengio Y., Haffner P., Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, **86**, (11), pp 2278–2324.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2010] LeCun Y., Kavukvuoglu K., Farabet C., Convolutional networks and applications in vision. In *International Symposium on Circuits and Systems*, pp 253–256.

[2015] LeCun Y., Bengio Y., Hinton G., Deep learning. *Nature*, 521, (7553), 436.

[1991] Lee Y., Oh S., Kim M., The effect of initial weights on premature saturation in back propagation learning. *Int Jt Conf Neural Netw*, **1**, pp 65–70.

[2004] Lee P.M., Bayesian statistics : An introduction. 3rd ed., Wiley.

[2005] Lee R.W., Implied volatility: Statistics, dynamics, and probabilistic interpretation. *Recent advances in applied probability*, Springer, New York, pp 241–268.

[2007a] Legg S., Hutter M., A collection of definitions of intelligence. *Frontiers in Artificial Intelligence and Applications*, volume 157, pp 17–24.

[2007b] Legg S., Hutter M., Universal intelligence: A definition of machine intelligence. *Minds and Machines*, **17**, (4), pp 391–444.

[2008] Lehman J., Stanley K.O., Exploiting open-endedness to solve problems through the search for novelty. In *Proceedings of Artificial Life XI*, volume 11, pp 329–336.

[2011a] Lehman J., Stanley K.O., Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, **19**, (2), pp 189–223.

[2011b] Lehman J., Stanley K.O., Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp 211–218. ACM.

[2006] Lehti P., Unsupervised duplicate detection using sample non-duplicates. PhD Thesis, Universitat Darmstadt.

[2015] Lei T., Barzilay R., Jaakkola T., Molding CNNs for text: Non-linear, non-consecutive convolutions. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp 1565-1575.

[2001] Lemarechal C., Lagrangian relaxation. In Michael Junger and Denis Naddef *Computational combinatorial optimization: Papers from the Spring School held in Schloss Dagstuhl*, May 15-19, 2000. Lecture Notes in Computer Science. Berlin, Springer-Verlag, pp 112–156.

[2012] Le Roux N., Bengio Y., Fitzgibbon A., Improving first and second-order methods by modeling uncertainty. In Sra, Suvrit; Nowozin, Sebastian; Wright, Stephen *J. Optimization for Machine Learning*, MIT Press, pp 404.

[1999] Levelt W.J.M., Models of word production. *Trends in Cognitive Sciences*, **3**, (6), pp 223–232

[1966] Levenshtein V.I., Binary codes capable of correcting deletions, insertions and reversals. Doklady Akademii Nauk SSSR, **163**, (4), pp 845–848, 1965, original in Russian-translation in Soviet Physics Doklady, **10**, (8), pp 707–710.

[2017] Levine S., Gupta A., Achiam J., Deep reinforcement learning. Berkeley, CS 294-112.

[2006] Lew T.L., Spencer A.B., Scarpa F., Worden K., Rutherford A., Hemez F., Identification of response surface models using genetic programming. *Mechanical Systems and Signal Processing*, **20**, (8), pp 1819–1831.

[2020] Lezcano C., Vazquez Noguera J.L., Pinto-Roa D.P., Garcia-Torres M., Gaona C., Gardel-Sotomayor P.E., A multi-objective approach for designing optimized operation sequence on binary image processing. Heliyon, Volume 6, Issue 4, e03670.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2014] Li B.G., Yu Z.G., Zhou Y., *J. Stat. Mech: Theor. Exp.*, **2**, P02020.

[2017] Li F-F., Johnson J., Yeung S., Lectures. Stanford Computer Science, Stanford.

[2010] Liao Z., Wang J., Forecasting model of global stock index by stochastic time effective neural network. *Expert Systems with Applications*, **37**, (1), pp 834–841.

[2016] Lillicrap T.P., Hunt J.J., Pritzel A., Heess N., Erez T., Tassa Y., Silver D., Wierstra D., Continuous control with deep reinforcement learning: Continuous Q-learning with actor network for approximate maximisation. Published as a conference paper at ICLR 2016.

[1993a] Lillo W.E., Loh M.H., Hui S., Zak S.H., On solving constrained optimisation problems with neural networks : A penalty method approach. *IEEE Transactions on Neural Networks*, **4**, (6), pp 931–940.

[1993b] Lillo W.E., Hui S., Zak S.H., Neural network for constrained optimization problems. *International Journal of Circuit Theory and Applications*, **21**, pp 385–399.

[1993] Lin L-J., Reinforcement learning for robots using neural networks. Degree of Doctor of Philosophy, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

[2008] Lin H-T., Li L., Support vector machinery for infinite ensemble learning. *Journal of Machine Learning Research*, **9**, pp 285–312.

[2009] Lin X., Yang Z., Song Y., Short-term stock price prediction based on echo state networks. *Expert Systems with Applications*, **36**, pp 7313–7317.

[2020] Lin T-Y., Focal Loss for Dense Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **42**, (2), pp 318–327. arXiv:1708.02002.

[2015] Ling W., etc. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of 2015 EMNLP*.

[1965] Lintner J., The valuation of risk assets and the selection of risky investments in stock portfolio and capital budgets. *Review of Economics and Statistics*, **47**, No. 1, pp 13–37.

[2002] Lipton A., The volatility smile problem. *Risk*, **2**, (2).

[1992] Littlestone N., Warmuth M.K., The weighted majority algorithm. University of California, Santa Cruz.

[1994] Littlestone N., Warmuth M., The weighted majority algorithm. *Info. and Computation*, **108**, (2), pp 212–261.

[2002] Littman M., Sutton R.S., Singh S., Predictive representations of state. *Advances in Neural Information Processing Systems 14*, Cambridge, MA, MIT Press.

[2002] Liu J., Lampinen J., On setting the control parameter of the differential evolution method. In *Proc. 8th Int. Conf. Soft Computing*, pp 11–18, Brno University of Technology, Czech Republic.

[2005] Liu J., Lampinen J., A fuzzy adaptive differential evolution algorithm. *Soft Computing*, **9**, (6), pp 448–462.

[2008] Liu D.H., A survey of planning in intelligent agents: From externally motivated to internally motivated systems. Technical Report TR-2008-936, Department of Computer Science, University of Rochester.

[2009] Liu B., Fernandez F.V., De Jonghe D., Gielena G., Less expensive and high quality stopping criteria for MC-based analog IC yield optimization. Working Paper, ESAT-MICAS, Katholieke Universiteit Leuven and IMSE, CSIC and University of Sevilla.

[2009b] Liu L., Ozsu T.M., Encyclopedia of database systems. Springer US.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2015] Liu J.L., Yu Z.G., Anh V., Determination of multifractal dimension of complex network by means of the sandbox algorithm. *Chaos*, **25**, 023103.

[2016] Liu P., Qiu X., Huang X., Modelling interaction of sentence pair with coupled-LSTMs. arXiv:1605.05573v2.

[2018] Liu P.J., Saleh M., Pot E., Goodrich B., Sepassi R., Kaiser L., Shazeer N., Generating wikipedia by summarizing long sequences. In *ICLR*.

[2018b] Liu W., Luo W., Lian D., Gao S., Future frame prediction for anomaly detection: A new baseline. In *CVPR*, pp 6536–6545.

[2018c] Liu B., Zhang T., Han F.X., Niu D., Lai K., Xu Y., Matching natural language sentences with hierarchical sentence factorization. arXiv:1803.00179v1.

[2019] Liu S., Oosterlee C.W., Bohte S.M., Pricing options and computing implied volatilities using neural networks. Working Paper.

[2019b] Liu N.F., Gardner M., Belinkov Y., Peters M.E., Smith N.A., Linguistic knowledge and transferability of contextual representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Volume 1, pp 1073–1094.

[2019c] Liu X., He P., Chen W., Gao J., Multi-Task Deep Neural Networks for Natural Language Understanding Working Paper,

[2019d] Liu Y., etc. RoBERTa: A robustly optimized BERT pretraining approach. Working Paper, arXiv:1907.11692.

[2020] Liu L., Jiang H., He P., Chen W., Liu X., Gao J., Han J., On the variance of the adaptive learning rate and beyond. Working Paper, arXiv:1908.03265v3.

[1991] Lo A.W., Long-term memory in stock market prices. *Econometrica*, **59**, (5), pp 1279–1313.

[2008] Lo A.W., Hedge funds, systemic risk, and the financial crisis of 2007-2008. Written Testimony of A.W. Lo, Prepared for the US House of Representative.

[2008] Lo A.W., Patel P.N., 130/30: The new long-only. *The Journal of Portfolio Management*, pp 12–38.

[2010] Lo A.W., Hasanhodzic J., The evolution of technical analysis: Financial prediction from Babylonian tablets to Bloomberg terminals. John Wiley and Sons, USA.

[2007] Long P.M., Servedio R.A., Random classification noise defeats all convex potential boosters. Working Paper. Published in *Machine Learning*, **78**, (3), pp 287–304, 2010.

[2015] Long J., Shelhamer E., Darrell T., Fully convolutional networks for semantic segmentation. In *CVPR*, arXiv:1411.4038v2.

[1995a] Longerstaey J., Zangari P., Five questions about RiskMetrics. Morgan Guaranty Trust Company, Market Risk Research, JPMorgan.

[1995b] Longerstaey J., More L., Introduction to RiskMetrics. 4th edition, Morgan Guaranty Trust Company, New York.

[1996] Longerstaey J., Spencer M., RiskMetrics: Technical document. Fourth Edition, Morgan Guaranty Trust Company, New York.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1996] Longin F., The asymptotic distribution of extreme stock market returns. *Journal of Business*, **69**, pp 383–408.

[2001] Longstaff F.A., Schwartz E.S., Valuing American options by simulation: A simple least-square approach. *The Review of Financial Studies*, **14**, (1), pp 113–147.

[1994] Loretan M., Phillips P., Testing the covariance stationarity of heavy-tailed time series. *Journal of Empirical Finance*, **1**, pp 211–248.

[2017] Loshchilov I., Hutter F., Fixing weight decay regularization in Adam. arXiv preprint arXiv:1711.05101.

[2009] Lu C-J., Lee T-S., Chiu C-C., Financial time series forecasting using independent component analysis and support vector regression. *Decision Support Systems*, **47**, (2), pp 115–125.

[2011] Lu H., Plataniotis K.N., Venetsanopoulos A.N., A survey of multilinear subspace learning for tensor data. *Pattern Recognition*, **44**, (7), pp 1540–1551.

[2017] Lu L., Zheng Y., Carneiro G., Yang L., Deep learning and convolutional neural networks for medical image computing. Springer.

[2019] Lu F., Deng J., Li S., Deng H., A hybrid of Differential Evolution and Genetic Algorithm for the multiple geographical feature label placement problem. The Special Issue Smart Cartography for Big Data Solutions, School of Geosciences and Info-Physics, Central South University, Changsha 410083, China.

[2006] Lu Dang Khoa N., Sakakibara K., Nishikawa I., Stock price forecasting using back propagation neural networks with time and profit based adjusted weight factors. *SICE-ICASE International Joint Conference*, Oct., Bexco, Busa, Korea, pp 5484–5488.

[1990] Luede E., Optmization of circuits with a large number of parameters. *Archiv f. Elektr. u. Uebertr.*, Band (44), Heft (2), pp 131-138.

[1984] Luenberger D.G., Linear and nonlinear programming. Addison-Wesley.

[1988] Lukac L., Brorsen B., Irwin S., A test of futures market disequilibrium using twelve different technical trading systems. *Applied Economics*, **20**, (5), pp 623–639.

[2006] Lukosevicius M., Popovici D., Jaeger H., Siewert U., Time warping invariant echo state networks Technical Report No. 2, Jacobs University Bremen.

[2007] Lukosevicius M., Echo state networks with trained feedbacks. Technical Report No. 4, Jacobs University Bremen.

[2009] Lukosevicius M., Jaeger H., Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, **3**, (3), pp 127–149.

[2010] Lukosevicius M., On self-organizing reservoirs and their hierarchies. Jacobs University Bremen, Tech. Rep, (25).

[2012] Lukosevicius M., A practical guide to applying echo state networks. Jacobs University Bremen, Germany.

[2012b] Lukosevicius M., Reservoir computing and self-organized neural hierarchies. PhD thesis, Jacobs University Bremen, Germany.

[2012] Lukosevicius M., Jaeger H., Schrauwen B., Reservoir computing trends. *Kunstliche Intelligenz*, Springer, **26**, (4), pp 365–371.

[2015] Luong M-T., Pham H., Manning C.D., Effective approaches to attention-based neural machine translation. Working Paper, arXiv:1508.04025v5.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2017] Lusch B., Brunton S.L., Kutz J.N., Deep learning for universal linear embeddings of nonlinear dynamics. arXiv:1712.09707.

[1996] Lux T., The stable Paretian hypothesis and the frequency of large returns: An examination of major German stocks. *Applied Economics Letters*, **6**, pp 463–475.

[2003] Lux T., The multi-fractal model of asset returns: Its estimation via GMM and its use for volatility forecasting. Economics Working Paper No. 2003-13, Christian-Albrechts-Universitat Kiel.

[2012] Lye C-T., Hooy C-W., Multifractality and efficiency: Evidence from Malaysian sectoral indices. *Int. Journal of Economics and Management*, **6**, (2), pp 278–294.

[1994] Ma J., Protter P., Yong J., Solving forward-backward stochastic differential equations explicitly: A four step scheme. *Probability theory and related fields*, **98**, pp 339–359.

[1998] Maass W., Orponen P., On the effect of analog noise in discrete-time analog computations. *Neural Computation*, **10**, pp 1071–1095.

[2002] Maass W., Natschlager T., Markram H., Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Computation*, **14**, (11), pp 2531–2560.

[2005a] Maass W., Joshi P., Sontag E.D., Computational aspects of feedback in neural circuits. Working Paper, It was published in 2007 in *PLoS Computational Biology*.

[2005b] Maass W., Joshi P., Sontag E.D., Principles of real-time computing with feedback applied to cortical microcircuit models. Conference Paper in *Advances in neural information processing systems*, January.

[2003] MacKay D.J.C., Information theory, inference, and learning algorithms. Cambridge University Press.

[1977] Mackey M.C., Glass L., Oscillation and chaos in physiological control systems. *Science*, **197**, (4300), pp 287–289.

[2009] Maei H., Szepesvari C., Bhatnagar S., Precup D., Silver D., Sutton R., Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems*, **22**, pp 1204–1212.

[1993] Maheswaran S., Sims C., Empirical implications of arbitrage-free asset markets. in P.C.B. Phillips, ed., *Models, Methods and Applications of Econometrics*, Cambridge, Basil Blackwell, pp 301–316.

[1987] Majani B.E., Decomposition methods for medium-term planning and budgeting. In S. Makridakis, S. Wheelwright, edt., *The handbook of forecasting: A manager's guide*, Whiley, New York, pp 219–237.

[2014] Makhzani A., Frey B., K-sparse autoencoders. In ICLR.

[2011] Maknickiene N., Vytautas Rutkauskas A., Maknickas A., Investigation of financial market prediction by recurrent neural network. *Innovative Infotechmologies for Science, Bussiness and Education*, **2**, (11), pp 3–8.

[1982] Makridakis S., Andersen A., Carbon R., Fildes R., Hibon M., Lewandowski R., Newton J., Parzen R., Winkler R., The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*, **1**, pp 111–153.

[1989] Mallat S.G., A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **11**, (7), pp 674–693.

[1990] Mallat S.G., Hwang W.L., Singularity detection and processing with wavelets. Technical Report No. 549, Computer Science Department, New York University.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1992] Mallat S.G., Hwang W.L., Singularity detection and processing with wavelets. *IEEE Trans. Inform. Theory*, **38**, pp 617–643.

[1992b] Mallat S.G., Zhong S., Complete signal representation with multiscale edges. *IEEE Trans.*, PAMI **14**, pp 710–732.

[1993] Malliaris M., Salchenberger L., A neural network model for estimating option prices. *Journal of Applied Intelligence*, , pp 193–206.

[1960] Mandelbrot B.B., The Pareto-Levy law and the distribution of income. *International Economic Revue*, **1**.

[1963a] Mandelbrot B.B., New methods in statistical economics. *The Journal of Political Economy*, **71**.

[1963] Mandelbrot B.B., The variation of certain speculative prices. *Journal of Business*, **36**, (4), pp 394–419.

[1968] Mandelbrot B.B., van Ness J., Fractional Brownian motions, fractional noises and applications. *SIAM Review*, **10**, (4), pp 422–437.

[1971] Mandelbrot B.B., When can price be arbitraged efficiently? A limit to the validity of the random walk and martingale models. *Re. Econom. Statist.*, **53**, pp 225–236.

[1974] Mandelbrot B.B., Intermittent turbulence in self similar cascades: Divergence of high moments and dimension of the carrier. *Journal of Fluid Mechanics*, **62**, (2), pp 331–358.

[1975] Mandelbrot B.B., Les objets fractals: forme, hasard et dimension. Paris, Flammarion.

[1979] Mandelbrot B.B., Taqqu M., Robust R/S analysis of long-run serial correlation. in *Proceedings of the 42nd Session of the International Statistical Institute*, , pp 69–104, Manila, Bulletin of the I.S.I..

[1982] Mandelbrot B.B., The fractal geometry of nature. W.H. Freeman and Company, New York.

[1997] Mandelbrot B.B., Fisher A., Calvet L., A multifractal model of asset returns. Cowles Foundation Discussion Paper No. 1164.

[2004] Mandelbrot B.B., The (mis)behavior of markets, a fractal view of risk, ruin and reward. Basic Books.

[2005] Manimaran P., Panigrahi P.K., Parikh J.C., Wavelet analysis and scaling properties of time series. *Phys. Rev. E*, **72**, 046120, pp 1–5.

[2009] Manimaran P., Panigrahi P.K., Parikh J.C., Multiresolution analysis of fluctuations in non-stationary time series through descrete wavelets. *Physica A*, **388**, pp 2306–2314.

[1995] Mann P.S., Introductory statistics. 2nd ed., Wiley.

[2000] Mantegna R.N., Stanley H.E., An introduction to econophysics: Correlation and complexity in finance. Cambridge University Press, Cambridge.

[1998] Marbach P., Smulated-based methods for Markov decision processes. Doctoral dissertation, MIT.

[1998] Marbach P., Tsitsiklis J.N., Smulated-based optimization of Markov reward processes. Technical Report, LIDS-P-2411, MIT.

[2018] Mardt A., Pasquali L., Wu H., Noe F., Vampnets for deep learning of molecular kinetics. *Nat. Commun.*, 9(1),5.

[2006] Mares M., Fuzzy Sets. *Scholarpedia*, **1**, (10), 2031.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1987] Margolus N.H., Physics and computation. PhD thesis, MIT.

[1952] Markowitz H.M., Portfolio selection. *Journal of Finance*, **7**, (1), pp 77–91.

[1952b] Markowitz H.M., The utility of wealth. *Journal of Political Economy*, **60**, pp 151–158.

[1959] Markowitz H.M., Portfolio selection, efficient diversification of investments. John Wiley and Sons, New York.

[2000] Mason L., Baxter J., Bartlett P., Frean M., Boosting algorithms as gradient descent. In S.A. Solla, T.K. Leen, and K-R. Muller, editors, *Advances in Neural Information Processing Systems 12*, pp 512–518, MIT Press.

[2002] Mason S.J., Graham N.E., Areas beneath the relative operating characteristics (ROC) and relative operating levels (ROL) curves: Statistical significance and interpretation. *Quarterly Journal of the Royal Meteorological Society*, **128**, (584), pp 2145–2166.

[2011] Marin J.M., Mengersen K., Robert C.P., Bayesian modelling and inference on mixtures of distributions. In Dey, D.; Rao, C.R. (eds.), Essential Bayesian models. Handbook of statistics: Bayesian thinking - modeling and computation. Elsevier.

[2012] Martens J., Sutskever I., Training deep and recurrent networks with hessian-free optimization. In *Neural networks: Tricks of the trade*, pp 1–58.

[2017] Marti L., Segredo E., Sanchez-Pi N., Hart E., Impact of selection methods on the diversity of many-objective Pareto set approximation. In *21st International Conference on Knowledge Based and Intelligent Information and Engineering Systems, KES2017*, 6-8 September 2017, Marseille, France.

[2003] Matia K., Ashkenazy Y., Stanley H.E., Multifractal properties of price fluctuations of stocks and commodities. *Europhysics Letters*, **61**, (3), pp 422–428.

[2004] Matos J.A.O., Gama S.M.A., Ruskin H., Duarte J., An econophysics approach to the Portuguese stock index-psi-20. *Physica A*, **342**, (3-4), pp 665–676.

[2008] Matos J.A.O., Gama S.M.A., Ruskin H.J., Al Sharkasi A., Crane M., Time and scale Hurst exponent analysis for financial markets. *Physica A*, **387**, pp 3910–3915.

[2006] Mausser H., Ding Y., Gregov S., Normalization and other topics in multi objective optimization. In *Proceedings of the Fields-MITACS Industrial Problems Workshop*, 2006.

[1994] Maxwell S.R., Experiments with a coroutine model for genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, Orlando, Florida, USA, volume 1, pp 413–417. IEEE Press.

[2007] Maydanchik A., Data quality assessment. Bradley Beach, New Jersey, Technics Publications.

[2004] McCallum A., Wellner B., Conditional models of identity uncertainty with application to noun coreference. *Advances in Neural Information Processing Systems (NIPS '04)*.

[2017] McCann B., Bradbury J., Xiong C., Socher R., Learned in translation: Contextualized word vectors. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

[2015] McClelland J.L., Exploration in parallel distributed processing: A handbook of models, programs, and exercises. Second Edition, Printer-Friendly PDF Version.

[2002] McCullagh P., What is a statistical model?, *Annals of Statistics*, **30**, (5), pp 1225–1310.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1943] McCulloch W., Pitts W., A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, (4), pp 115–133.

[1996] McCulloch J.H., Financial applications of stable distributions. in G.S. Maddala, C.R. Rao, eds., *Handbook of Statistics*, **14**, Elsevier, pp 393–425.

[1997] McCulloch J.H., Measuring tail thickness to estimate the stable index α: A critique. *Journal of Business & Economic Statistics*, **15**, (1), pp 74–81.

[2015] McElreath R., Statistical rethinking. First Edition. Chapman and Hall

[2018] McGhee W.A., An artificial neural network representation of the SABR stochastic volatility model. Preprint.

[2008] McGilvray D., Executing data quality projects: Ten steps to quality data and trusted information. San Francisco, CA, Morgan Kaufmann Publishers Inc..

[2008] McKay B., Evolutionary Algorithms. Encyclopedia of Ecology, Seoul National University, Seoul, Republic of Korea, pp 1464–1472.

[2004] McLachlan G.J., Discriminant analysis and statistical pattern recognition. Wiley Interscience.

[1998] McPhee N.F., Hopper N.J., Reierson M.L., Sutherland: An extensible object-oriented software framework for evolutionary computation. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, Genetic Programming 1998: *Proceedings of the Third Annual Conference*, pp 241, University of Wisconsin, Madison, Wisconsin, USA. Morgan Kaufmann.

[1989] Mendelzon A.O., Wood P.T., Finding regular simple paths in graph databases. In *Proceedings of the 15th International Conference on Very Large Data Bases (VLDB)*, Morgan Kaufmann Publishers Inc., pp 185–193.

[2010] Menkhoff L., The use of technical analysis by fund managers: International evidence. *Journal of Banking & Finance*, **34**, (11), pp 2573–2586.

[2004] Menon A., Frontiers of evolutionary computation. Kluwer Academic Publishers.

[1971] Merton R.C., Optimum consumption and portfolio rules in a continuous-time model. *Journal of Economic Theory*, **3**, pp 373–413.

[1981] Merton R.C., On market timing and investment oerformance: An equilibrium theory of value for market forecasts. *Journal of Business*, **54** (3), pp 363–406.

[2007] Metghalchi M., Glasure Y., Garza-Gomez X., Chen C., Profitable technical trading rules for the Austrian stock market. *International Business & Economics Research Journal*, **6**, (9), pp 49–58.

[2012] Metghalchi M., Marcucci J., Chang Y-H., Are moving average trading rules profitable? Evidence from the European stock markets. *Applied Economics*, **44**, (12), pp 1239–1559.

[2015] Metghalchi M., Chen C., Hayes L.A., History of share prices and market efficiency of the Madrid general stock index. *International Review of Financial Analysis*, **40**, pp 178–184

[2018] Metghalchi M., Hajilee M., Hayes L.A., Return predictability and efficient market hypothesis: Evidence from Iceland. *The Journal of Alternative Investments*, **21**, (1), pp 68–78.

[2003] Meyer J-A., Doncieux S., Guillot D., Guillot A., Evolutionary approaches to neural control of rolling, walking, swimming and flying animats or robots. *Biologically Inspired Robot Behavior Engineering*, pp 1–43.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2004a] Mezura-Montes E., Coello Coello C.A., Tun-Morales E.I., Simple feassibility rules and differential evolution for constrained optimization. *Third Mexican International Conference on Artificial Intelligence, MICAI, Lecture Notes in Artificial Intelligence*, pp 707–716.

[2004] Mezura-Montes E., Coello Coello C.A., A study of mechanisms to handle constraints in evolutionary algorithms. Workshop at the Genetic and Evolutionary Computation Conference, Seattle, Washington, ISGEC.

[2006] Mezura-Montes E., Velazquez-Reyes J., Coello Coello C.A., Modified differential evolution for constrained optimization. *IEEE Congress on Evolutionary Computation*, IEEE Press, pp 332–339.

[2006b] Mezura-Montes E., Coello Coello C.A., Velazquez-Reyes J., Munoz-Davila L., Multiple offspring in differential evolution for engineering design. *Engineering Optimization*, **00**, pp 1–33.

[1986] Micchelli C.A., Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, **2**, pp 11–22.

[1995] Michalewicz Z., Gnetic algorithms, numerical optimization and constraints. L. Eshelman, ed., *Proceeding of the Sixth International Conference on Genetic Algorithms*, San Mateo, pp 151–158.

[1996] Michalewicz Z., Genetic algorithms + data structures = evolution programs. Springer Verlag.

[2013] Michalski R.S., Carbonell J.G., Mitchell T.M., Machine learning: An artificial intelligence approach. Springer Science & Business Media.

[2006] Miikkulainen R., Bryant B.D., Cornelius R., Karpov I.V., Stanley K.O., Yong C.H., Computational intelligence in games. In Gary Y. Yen and David B. Fogel, editors, *Computational Intelligence: Principles and Practice*, IEEE Computational Intelligence Society, Piscataway, NJ.

[2010] Mikolov T., Karafiat M., Burget L., Cernocky J., Khudanpur S., Recurrent neural network based language model. In *INTERSPEECH*, pp 1045–1048.

[2012] Mikolov T., Statistical language models based on neural networks. PhD thesis, Brno University of Technology.

[2013] Mikolov T., Chen K., Corrado G., Dean J., Efficient estimation of word representations in vector space. Working Paper, arXiv:1301.3781.

[2013b] Mikolov T., Sutskever I., Chen K., Corrado G., Dean J., Distributed representations of words and phrases and their compositionality. Working Paper, Google.

[2003] Mikosch T., Starica C., Long-range dependence effects and ARCH modeling. In *Theory and Applications of Long-range Dependence*, Birkhauser Boston, Boston, pp 439–459.

[1990] Miller G.A., Beckwith R., Fellbaum C.D., Gross D., Miller K., WordNet: An online lexical database. *Int. J. Lexicograph*, **3**, 4, pp 235–244.

[1996] Miller B.L., Shaw M.J., Genetic algorithms with dynamic niche sharing for multimodal function optimization. In *Proceedings of IEEE International Conference on Evolutionary Computing*, Nagoya, Japan.

[1999] Miller J.F., An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pp 1135–1142, Orlando, Florida, USA. Morgan Kaufmann.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2006] Miller J.F., Smith S.L., Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, **10**, (2), pp 167–174.

[2018] Miller J., Recurrent models don't need to be recurrent. Berkeley Artificial Intelligence Research.

[2019] Miller J., Hardt M., Stable recurrent models. arXiv:1805.10369.

[1974] Milstein G.N., Approximate integration of stochastic differential equations. *Teoriya Veroyatnostei i ee Primeneniya (in Russian)*, **19**, (3), pp 583–588.

[2009] Mingguang L., Gaoyang L., Artificial neural network co-optimisation algorithm based on differential evolution. In *Second International Symposium on Computational Intelligence and Design*, pp 256–559.

[1963] Minsky M., Steps toward artificial intelligence. In *Proceedings of the Institute of Radio Engineers*, 1961, **49**, pp 8–30. Reprinted in *Computers and Thought*, ed. by E.A. Feigenbaum and J. Feldman, New York, McGraw-Hill, pp 406–450.

[1967] Minsky M., Computation: Finite and infinite machines. Prentice-Hall, Englewood Cliffs, (First ed.).

[1969] Minsky M.L., Papert S.A., Perceptrons: An introduction to computational geometry. The MIT Press, Cambridge, MA.

[2014] Mirza M., Osindero S., Conditional generative adversarial networks. Working Paper, arXiv:1411.1784.

[2016] Misra I., Lawrence Zitnick C., Hebert M., Shuffle and learn: Unsupervised learning using temporal order verification. Working Paper, arXiv:1603.08561v2.

[1997] Mitchell T.M., Machine learning. McGraw-Hill. Paperback version in 2017.

[2008] Mitchell J., Lapata M., Vector-based models of semantic composition. In *ACL*, pp 236–244.

[2003] Mittelmann H.D., An independent benchmarking of SDP and SOCP solvers. *Mathematical Programming*, **95**, (2), pp 407–430.

[1999] Mladenov V.M., Maratos N.G., Neural networks for solving constrained optimization problems. Working Paper, Technical University of Sofia.

[2014] Mnih V., Heess N., Graves A., Kavukcuoglu K., Recurrent models of visual attention. Working Paper, arXiv:1406.6247v1.

[2015] Mnih V., Kavukcuoglu K., Silver D., Rusu A.A., Veness J., Bellemare M.G., Graves A., Riedmiller M., Fidjeland A.K., Ostrovski G., etc. Human-level control through deep reinforcement learning: Q-learning with convolutional networks for playing Atari. First seen in NIPS DL Workshop 2013, arXiv:1312.5602. In *Nature*, **518**, pp 529–533.

[2016] Mnih V., Badia A.P., Mirza M., Graves A., Lillicrap T.P., Harley T., Silver D., Kavukcuoglu K., Asynchronous methods for deep reinforcement earning. arXiv preprint arXiv:1605.03835.

[2015] Mohamed S., A statistical view of deep learning. Working Paper.

[2016] Mohamed S., Lakshminarayanan B., Learning in implicit generative models. arXiv:1610.03483.

[2012] Mohri M., Rostamizadeh A., Talwalkar A., Foundations of machine learning. MIT Press, Massachusetts, USA. Second Edition was out in 2018.

[1996] Monge A.E., Elkan C.P., The field matching problem: Algorithms and applications. In *Proceedings of the AAAI International Conference on Knowledge Discovery and Data Mining AAAI*.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1997] Monge A.E., Elkan C.P., An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proc. Second ACM SIGMOD Workshop Research Issues in Data Mining and Knowledge Discovery (DMKD '97)*, pp 23–29.

[1989] Montana D.J., Davis L., Training feedforward neural networks using genetic algorithms. *Int Jt Conf Artif Intell*, **89**, pp 762–767.

[1995] Montana D.J., Strongly typed genetic programming. *Evolutionary Computation*, **3**, (2), pp 199–230.

[2012] Montavon G., Orr G., Muller K-R., Neural networks: Tricks of the trade. Second edition, Lectures Notes Computer State-of-the-Art Surveys, Springer.

[1998] Moravec H., When will computer hardware match the human brain? *Journal of Evolution and Technology*, **1**, 12 pages.

[2008] Moreau E., Yvon F., Cappe O., Robust similarity measures for named entities matching. In *Proceedings of the International Conference on Computational Linguistics ACL.*

[2007] Moret M.A., Zebende G.F., Amino acid hydrophobicity and accessible surface area. *Phys. Rev. E*, **75**, 011920.

[1994] Moriarty D.E., Miikkulainen R., Efficient reinforcement learning through symbiotic evolution. Technical Report AI94-224, The University of Texas at Austin, Department of Computer Sciences.

[2016] Mou L., et al. Natural language inference by tree-based convolution and heuristic matching. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics ACL.*

[2008] Mouret J-B., Doncieux S., Incremental evolution of animats' behaviors as a multi-objective optimization. In *Proceedings of the 10th international conference on Simulation of Adaptive Behavior: From Animals to Animats*, pp 210–219. Springer-Verlag.

[2009a] Mouret J-B., Doncieux S., Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, pp 1161–1168, Institute of Electrical and Electronics Engineers Inc..

[2009b] Mouret J-B., Doncieux S., Using behavioral exploration objectives to solve deceptive problems in neuro-evolution. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pp 627–634. ACM.

[2011] Mouret J-B., Novelty-based multiobjectivization. In *New horizons in evolutionary robotics*, pp 139–154.

[2012] Mouret J-B., Doncieux S., Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary Computation*, **1**, (20).

[2015] Mouret J-B., Clune J., Illuminating search spaces by mapping elites. Working Paper.

[2006] Moyano L.G., de Souza J., Duarte Queiros S.M., Multi-fractal structure of traded volume in financial markets. *Physica A*, **371**, pp 118–121.

[1989] Mozer M., Smolensky P., Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D. Touretzky, editor, *Advances in Neural Information Processing*, Systems 1, pp 107–115. Morgan Kaufmann, San Mateo, CA.

[2016] Mueller J., Thyagarajan A., Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the AAAI Conference on Artificial Intelligence AAAI*, pp 2786–2792.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1993] Muhlenbein H., Schlierkamp-Voosen D., Predictive models for the breeder genetic algorithm: I. continuous parameter optimization. *Evol. Comput.*, **1**, pp 25–49.

[1990] Muller U.A., Dacorogna M.M., Olsen R.B., Pictet O.V., Schwarz M., Morgenegg C., Statistical study of foreign exchange rates, empirical evidence of a price change scaling law, and intraday analysis. *Journal of Banking and Finance*, **14**, pp 1189–1208.

[1997] Muller U.A., Dacorogna M.M., Dave R.D., Olsen R.B., Pictet O.V., von Weizsacker J.E., Volatilities of different time resolutions: Analyzing the dynamics of market components. *Journal of Empirical Finance*, **4**, pp 213–239.

[2012] Muller H., Freytag J-C., Leser U., Improving data quality by source analysis. *J. Data and Information Quality*, **2**, (4), pp 15:1–15:38, Mar. 2012.

[2016] Munos R., Stepleton T., Harutyunyan A., Bellemare M.G., Safe and efficient off-policy reinforcement learning. Google DeepMind.

[2018] Murdoch W.J., Liu P.J., Yu B., Beyond word importance: Contextual decomposition to extract interactions from lstms. International Conference on Learning Representations.

[2009] Murguia J.S., Perez-Terrazas J.E., Rosu H.C., Multifractal properties of elementary cellular automata in a discret wavelet approach of MF-DFA. *EPL Journal*, **87**, pp 2803–2808.

[1999] Murphy J.J., Technical analysis of the financial markets: A comprehensive guide to trading methods and applications. Prentice Hall Press.

[2005] Murphy S.A., A generalization error for Q-learning. *Journal of Machine Learning Research*, **6**, pp 1073–1097.

[2012] Murphy K.P., Machine learning: A probabilistic perspective. Cambridge, MA, MIT Press.

[2003] Murtagh F., Stark J.L., Renaud O., On neuro-wavelet modeling. Working Paper, School of Computer Science, Queen's University Belfast.

[1991] Muzy J.F., Bacry E., Arneodo A., Wavelets and multifractal formalism for singular signals: Application to turbulence data. *Phys. Rev. Lett.*, **67**, (25), pp 3515–3518.

[1993] Muzy J.F., Bacry E., Arneodo A., Multifractal formalism for fractal signals: The structure-function approach versus the wavelet-transform modulus-maxima method. *Phys. Rev. E*, **47**, (2), pp 875–884.

[2011] Nair B.B., Sai S.G., Naveen A., Lakshmi A., Venkatesh G., Mohandas V., A GA-artificial neural network hybrid system for financial time series forecasting. *Inf Techno Mob Commun.*, **147**, pp 499–506.

[2014] Nanculef R., Frandi E., Sartori C., Allendea H., A novel Frank-Wolfe algorithm. Analysis and applications to large-scale SVM training *Inf. Sci.*, **285**, pp 66–99.

[2001] Navarro G., A guided tour to approximate string matching. *ACM Computing Surveys*, **33**, (1), pp 31–88.

[2015] Nebel B., Logics for knowledge representation. In International Encyclopedia of the Social & Behavioral Sciences (Second Edition).

[1970] Needleman S.B., Wunsch C.D., A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Molecular Biology*, **48**, (3), pp 443–453.

[2001] Neely C.J., Weller P.A., Technical analysis and central bank intervention. *Journal of International Money and Finance*, **20**, (7), pp 949–970.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2003] Neiling M., Jurk S., The object identification framework. In *Proceedings of the 2003 ACM SIGKDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pp 37–39, Washington, DC.

[1990] Nelson D.B., ARCH models as diffusion approximations. *Journal of Econometrics*, **45**, pp 7–38.

[1991] Nelson D.B., Conditional heteroskedasticity in asset returns: A new approach. *Econometrica*, **59**, pp 347–370.

[2011] Neumann K., Steil J.J., Batch intrinsic plasticity for extreme learning machines. In *International Conference on Artificial Neural Networks*, pp 339–346, Springer.

[1959] Newcombe H.B., Kennedy J.M., Axford S., James A., Automatic linkage of vital records. *Science*, **130**, (3381), pp 954–959.

[1999a] Newman M.E.J., Watts D.J., Renormalization group analysis of the small-world network model. *Phys. Letter A*, **263**, pp 341–346.

[1999b] Newman M.E.J., Watts D.J., Scaling and percolation in the small-world network model. *Physical Review E*, **60**, (6), 7332.

[2001] Newman M.E.J., The structure of scientific collaboration networks. *PNAS*, **98**, (2), pp 404–409.

[2000] Ng A.Y., Russell S.J., Algorithms for inverse reinforcement learning. In *Proceedings of the* 17*th International Conference on Machine Learning*, pp 663–670.

[2002] Ng A.Y., Jordan M,I., On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. Working Paper, University of California, Berkeley.

[2019] Ngo C.P., Winarto A.A., Li C.K.K., Park S., Akram F., Lee H.K., Fence GAN: Towards better anomaly detection. arXiv:1904.01209.

[2015] Nguyen A., Yosinski J., Clune J., Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

[2018] Nguyen D.Q., Billingsley R., Du L., Johnson M., Improving topic models with latent feature word representations. arXiv:1810.06306.

[2019] Nguyen D.Q., Verspoor K., End-to-end neural relation extraction using deep biaffine attention. In *Proceedings of the 41st European Conference on Information Retrieval (ECIR)*. arXiv:1812.11275

[2017] Nie Y., Bansal M., Shortcut-stacked sentence encoders for multi-domain inference. In *Proceedings of the Workshop on Evaluating Vector Space Representations for NLP*.

[2013] Niere H.M., A multifractality measure of stock market efficiency in Asean region. *European Journal of Business and Management*, **5**, (22), pp 13–19.

[2014] Niu H., Wang J., Financial time series prediction by a random data-time effective rbf neural network. *Soft Computing*, **18**, (3), pp 497–508.

[1999] Nocedal J., Wright S.J., Numerical optimization. Sringer Verlag.

[2001] Nolan J.P., Maximum likelihood estimation and diagnostics for stable distributions. in O.E. Barndorff-Nielsen, T. Mikosch, S. Resnick, eds., *Levy Processes*, Brikhauser, Boston.

[2014] Nolan J.P., Stable distributions: Models for heavy-tailed data. Math/Stat Department, American University.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1991] Nolfi S., Parisi D., Growing neural networks. Technical Report PCIA-91-15, Institute of Psychology.

[2011] Noman N., Bollegala D., Iba H., An adaptive differential evolution algorithm. *IEEE*, pp 2229–2236.

[2010] Nopiah Z.M., Khairir M.I., Abdullah S., Baharin M.N., Arifin A., Time complexity analysis of the Genetic Algorithm clustering method. Department of Mechanical and Materials Engineering Universiti Kebangsaan Malaysia.

[1994] Nordin P., A compiling genetic programming system that directly manipulates the machine code. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 14, pp 311–331. MIT Press.

[1997] Nordin P., Evolutionary program induction of binary machine code and its applications. PhD thesis, der Universitat Dortmund am Fachereich Informatik.

[1999] Nordin P., Banzhaf W., Francone F.D., Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. In Lee Spector, William B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 12, pp 275–299. MIT Press, Cambridge, MA, USA.

[2005] Norouzzadeh P., Jafari G.R., Application of multifractal measures to Tehran price index. *Physica A*, **356**, pp 609–627.

[2006] Norouzzadeh P., Rahmani B., A multifractal detrended fluctuation description of Iranian rial-US dollar exchange rate. *Physica A*, **367**, pp 328–336.

[2013] Nothman J., Ringland N., Radford W., Murphy T., Curran J.R., Learning multilingual named entity recognition from Wikipedia. *Artificial Intelligence*, **194**, pp 151–175.

[2012] Noto K., Brodley C., Slonim D., FRaC: A feature-modeling approach for semi-supervised and unsupervised anomaly detection. *Data mining and knowledge discovery*, **25**, (1), pp 109–133.

[2004] Oh K.S., Jung J., GPU implementation of neural networks. *Pattern Recognition*, **37**, (6), pp 1311–1314.

[2012] Oh G., Eom C., Havlin S., Jung W-S., Wang F., Stanley H.E., Kim S., A multifractal analysis of Asian foreign exchange markets. *European Physical Journal B*, pp 85–214.

[2015] O'hagan S., Kell D.B., Software review: The KNIME workflow environment and its applications in genetic programming and machine learning. *Genetic Programming and Evolvable Machines*, **16**, (3), pp 387–391.

[1993] Ohlsson M., Peterson C., Soderberg B., Neural networks for optimization problems with inequality constraints: The knapsack problem. *Neural Computation*, **5**, (2), pp 331–339.

[1998] Oksendal B., Stochastic differential equations. Springer Fifth Edition.

[2003] Olson D., Mossman C., Neural network forecasts of Canadian stock returns using accounting ratios. *International Journal of Forecasting*, **19**, pp 453–465.

[2003] O'Neill M., Ryan C., Grammatical evolution: Evolutionary automatic programming in a arbitrary language. volume 4 of Genetic programming. Kluwer Academic Publishers.

[2004] Onwubolu G.C., Differential evolution for the flow shop scheduling problem. in *New Optimization Techniques in Engineering*, G.C. Onwubolu and B. Babu, Eds., Springer-Verlag, Berlin, pp 585–611.

[2002] Ormoneit D., Sen S., Kernel-based reinforcement learning. *Machine Learning*, **49**, pp 161–178.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2013] Osborne J.W., Best practices in data cleaning: A complete guide to everything you need to do before and after collecting your data. SAGE Publications, Inc, First edition.

[2013] Fernando F.E.B., Freitas A.A., Rice j., Johnson C.G., New sequential covering strategy for inducing classification rules with ant colony algorithms. *IEEE Transactions on Evolutionary Computation*, **17**, (1), pp 64–76.

[2006] Ouyang Z., Stable process. Working Notes.

[2005] Oyama A., Shimoyama K., Fujii K., New constraint-handling method for multi-objective multi-constraint evolutionary optimization and its application to space plane design. *Evolutionary and Deterministic Methods for Design*.

[2020] Oyedotun O.K., Shabayek A.R., Aouda D., Ottersten B., Improved highway network block for training very deep neural networks. *IEEE Access*.

[2018] Pachon A., Sacerdote L., Yang S., Scale-free behavior of networks with the copresence of preferential and uniform attachment rules. *Physica D: Nonlinear Phenomena*, arXiv:1704.08597.

[1991] Padberg M., Rinaldi G., A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems *SIAM Rev.*, **33**, (1), pp 60–100.

[2002] Palmer A., Montano J.J., Redes neuronales artificiales aplicadas al analisis de datos. Doctoral Dissertation. University of Palma de Mallorca.

[2010] Pan S.J., Yang Q., A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, **22**, (10), pp 1345–1359.

[2019] Pan J., Chen K., Shi J., Feng H., Ouyang W., Lin D., Libra R-CNN: Towards balanced learning for object detection. arXiv:1904.02701v1

[2001] Panas E., Estimating fractal dimension using stable distributions and exploring long memory through ARFIMA models in Athens stock exchange. *Applied Financial Economics*, **11**.

[2020] Pang G., Shen C., Cao L., Van Den Hengel A., Deep learning for anomaly detection: A review. arXiv:2007.02500v2.

[1998] Papadimitriou C., Tamaki H., Raghavan P., Vempala S., Latent semantic indexing: A probabilistic analysis. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp 159–168.

[1998] Papageorgiou G., Likas A., Stafylopatis A., A hybrid neural optimization scheme based on parallel updates. *International Journal of Computer Mathematics*, **67**, pp 223–237.

[1990] Pardoux E., Peng S., Adapted solution of a backward stochastic differential equation. *System Control Lett.*, **14**, pp 55–61.

[1999] Pardoux E., Tang S., Forward-backward stochastic differential equations and quasilinear parabolic PDEs. *Probab Theor Relat Fields*, **114**, pp 123–150.

[1985] Parisi G., Frisch U., On the singularity structure of fully developed turbulence. in M. Ghil, R. Benzi, G. Parisi, eds., *Turbulence and Predictability in Geophysical Fluid Dynamics*, Proceedings of the International School of Physics, Amsterdam, pp 84–87.

[2020] Park D., Song H., Kim M., Lee J., TRAP: Two-level regularized autoencoder-based embedding for power-law distributed data. In *TheWebConf*.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1985] Parker D.B., Learning logic. Report TR-47, MIT Press.

[2011] Parker J.R., Algorithms for image processing and computer vision. Wiley, (2nd ed.).

[2002] Parkes A., Introduction to languages, machines and logic: Computable languages, abstract machines and formal logic. Springer.

[1997] Partridge D., Krzanowski W.J., Software diversity: Practical statistics for its measurement and exploitation. *Information & Software Technology*, **39**, (10), pp 707–717.

[2013] Pascanu R., Mikolov T., Bengio Y., On the difficulty of training recurrent neural networks. in *Proceedings of the 30th International Conference on Machine Learning*.

[1995] Pasemann F., Neuromodules: A dynamical systems approach to brain modelling. In Hans J. Herrmann, D. E. Wolf, and Ernst Poppel, editors, *Workshop on Supercomputing in Brain Research: from tomography to neural networks*, pp 21–23, World Scientific Publishing Co..

[1999] Pasemann F., Steinmetz U., Dieckman U., Evolving structure and function of neurocontrollers. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 3, pp 1973–1978, Washington D.C., USA. IEEE Press.

[2000] Pasquini M., Serva M., Clustering of volatility as a multiscale phenomenon. *European Physical Journal B*, **16**, (1), pp 195–201.

[2002] Pasula H., Marthi B., Milch B., Russell S.J., Shpitser I., Identity uncertainty and citation matching. *Advances in Neural Information Processing Systems (NIPS '02)*, pp 1401–1408.

[2015] Patel J., Shah S., Thakkar P., Kotecha K., Predicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques. *Expert Systems with Applications*, **42**, pp 259–268.

[1985] Pearl J., Bayesian networks: A model of self-activated memory for evidential reasoning. (UCLA Technical Report CSD-850017). In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California*, Irvine, CA., pp 329–334.

[2008a] Pedersen M.E.H., Chipper

eld A.J., Parameter tuning versus adaptation: Proof of principle study on differential evolution. Technical Report HL0802, Hvass Laboratories.

[2008b] Pedersen M.E.H., Chipper

eld A.J., Tuning differential evolution for artificial neural networks. Hvass Laboratories Technical Report no. HL0803.

[2010] Pedersen M.E.H., Tuning and simplifying heuristical optimization. PhD thesis, Computational Engineering and Design Group, School of Engineering Sciences, University of Southampton.

[1931] Peirce C.S., Collected papers. Harvard University Press.

[1994] Peng C.K., Buldyrev S.V., Havlin S., Simons M., Stanley H.E., Goldberger A.L., Mosaic organization of DNA nucleotides. *Physical Review E*, **49**, (2), pp 1685–1689.

[2017] Peng J., Agliari E., Scaling laws for diffusion on (trans)fractal scale-free networks. *Chaos*, **27**, (8), 083108.

[2014] Pennington J., Socher R., Manning C., Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp 1532–1543.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2019] Perera P., Nallapati R., Xiang B., OCGAN: One-class novelty detection using gans with constrained latent representations. In *CVPR*, pp 2898–2906.

[1994] Perry J.E., The effect of population enrichment in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, pp 456–461. IEEE Press.

[1991-96] Peters E.E., Chaos and order in the capital markets. second edition, John Wiley & Sons.

[1994] Peters E.E., Fractal market analysis: Applying chaos theory to investment and economics. John Wiley & Sons.

[2005] Peters J., Vijayakumar S., Schaal S., Natural actor-critic. In *16th European Conference on Machine Learning*, pp 280–291.

[2006] Peters J., Schaal S., Policy gradient methods for robotics. In *International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp 2219–2225.

[2008] Peters J., Schaal S., Reinforcement learning of motor skills with policy gradients. *Neural Networks*, **21**, (4), pp 682–697.

[2011c] Peters O., Optimal leverage from non-ergodicity. *Quantitative Finance*, **11** (11), pp 1593–1602.

[2018] Peters M.E., Neumann M., Iyyer M., Gardner M., Clark C., Lee K., Zettlemoyer L., Deep contextualized word representations. In *Proceedings of 2018 NAACL*, volume 1, pp 2227–2237.

[2019] Peters M.E., Ruder S., Smith N.A., To tune or not to tune? Adapting pretrained representations to diverse tasks. Working Paper, arXiv:1903.05987v2.

[2006] Pezier J., White A., The relative merits of investable hedge fund indices and of funds of hedge funds in optimal passive portfolios. ICMA Centre Discussion Papers in Finance, The University of Reading.

[1998] Pfister G., In search of clusters. Upper Saddle River, Prentice Hall.

[2019] Phang J., Fevry T., Bowman S.R., Sentence encoders on STILTs: Supplementary training on intermediate labeled-data tasks. Working Paper, arXiv:1811.01088v2,

[1991] Piatetsky-Shapiro G., Discovery, analysis, and presentation of strong rules. In Piatetsky-Shapiro, Gregory and Frawley, William J., eds., Knowledge Discovery in Databases, AAAI/MIT Press, Cambridge, MA.

[1987] Pineda F.J., Generalization of backpropagation to recurrent and higher order neural networks. In *Proceedings of IEEE Conference on Neural Information Processing Systems*, Denver, Colorado.

[1970] Plotkin G.D., Automatic methods of inductive inference. PhD thesis, University of Edinburgh.

[1990] Poggio T., Girosi F., Networks for approximation and learning. Proceedings of ZEEE, **78**, pp 1481–1497.

[1892-99] Poincare H., New methods of celestial mechanics. 3 vols. English trans., 1967.

[1905-10] Poincare H., Lessons of celestial mechanics.

[1997] Polak E., Optimization: Algorithms and consistent approximations.

[1996] Poli R., Discovery of symbolic, neuro-symbolic and neural networks with parallel distributed genetic programming. Technical Report CSRP-96-14, University of Birmingham, School of Computer Science. Presented at 3rd International Conference on Artificial Neural Networks and Genetic Algorithms, ICANNGA 97.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1998] Poli R., Langdon W.B., Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, **6**, (3), pp 231–252.

[2005] Poli R., Langdon W.B., Running genetic programming backward. In Tina Yu, Rick L. Riolo, and Bill Worzel, editors, *Genetic Programming Theory and Practice III*, volume 9 of Genetic Programming, chapter 9, pp 125–140, Ann Arbor. Springer.

[2006] Poli R., Langdon W.B., Backward-chaining evolutionary algorithms. *Artificial Intelligence*, **170**, (11), pp 953–982.

[2007] Poli R., Langdon W.B., McPhee N.F., Koza J.R., Genetic programming an introductory tutorial and a survey of techniques and applications. Technical Report CES-475.

[2015] Polydoros A.S., Nalpantidis L., Kruger V., Advantages and limitations of reservoir computing on model learning for robot control. IROS Workshop on Machine Learning in Planning and Control of Robot Motion

[1993] Pomerleau D.A., Knowledge-based training of artificial neural networks for autonomous robot driving. In J. Connell and S. Mahadevan, (eds.), *Robot Learning*, pp 19–43, Kluwer Academic Publishers.

[1936] Post E., Finite combinatory processes, formulation I. *The Journal of Symbolic Logic*, **1**, (3), pp 103–105.

[2001] Potters M., Bouchaud J.P., Sestovic D., Hedged Monte Carlo: Low variance derivative pricing with objective probabilities. *Physica A*, **289**, pp 517–525.

[1987] Powell M.J.D., Radial basis functions for multivariable interpolation: A review. In J.C. Mason and M.G. Cox, eds. *Algorithms for Approximation*, Clarendon Press, Oxford, pp 143–167.

[2011] Powers D.M.W., Evaluation: From precision, recall and F-Measure to ROC, informedness, markedness & correlation. *Journal of Machine Learning Technologies*, **2**, (1), pp 37–63.

[1992] Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P., Numerical recipes. 2nd ed. Cambridge, Cambridge University Press.

[1976] Price D.J. de Solla, A general theory of bibliometric and other cumulative advantage processes. *Journal of American Society for Information Science*, **27**, (5), pp 292–306.

[2005] Price K., Storn R., Lampinen J., Differential evolution: A practical approach to global optimization. Springer, Heidelberg.

[2011] Prinz J-H., Wu H., Sarich M., Keller B., Senne M., Held M., Chodera J.D., Schutte C., Noe F., Markov models of molecular kinetics: Generation and validation. *J. Chem. Phys.*, **134**, (17), 174105.

[1994] Puterman M.L., Markov decision processes: Discrete stochastic dynamic programming. John Wiley & Sons.

[1999] Pyle D., Data preparation for data mining. Morgan Kaufmann Publishers.

[2001] Qi M., Predicting US recessions with leading indicators vianeural network models. *International Journal of Forecasting*, **17**, pp 383–401.

[2016] Qiu M., Song Y., Predicting the direction of stock market index movement using an optimized artificial neural network model. PLoS ONE **11** ,(5): e0155133. doi:10.1371/journal.

[1993] Quinlan J.R., C4.5: Programs for machine Learning. Morgan Kaufmann, San Francisco, CA.

[1998] Quinlan J.R., Induction of decision trees. *Machine Learning*, **1**, (1), pp 81–106.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1993] Rabemananjara R., Zakoian J., Threshold ARCH models and asymmetries in volatility. *Journal of Applied Econometrics*, **8**, pp 31–49.

[1986] Rabiner L., Juang B.H., An introduction to hidden Markov models. *IEEE Acoust. Speech Sign. Process. Mag.*, **3**, pp 4–16.

[1989] Rabiner L., A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, **77**, (2), pp 257–286.

[2015] Radford A., Metz L., Chintala S., Unsupervised representation learning with deep convolutional generative adversarial networks. Working Paper, arXiv:1511.06434.

[2018] Radford A., Narasimhan K., Salimans T., Sutskever I., Improving language understanding by generative pretraining. Working Paper.

[2019] Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I., Language models are unsupervised multitask learners. Working Paper.

[2016] Raffel C., Ellis D.P.W., Feed-forward networks with attention can solve some long-term memory problems. arXiv:1512.08756v5.

[1985] Raftery A.E., A model for high-order Markov chains. *J. R. Statist. Soc. B*, pp 528–539.

[2020] Raghavan A., Using machine learning in data quality management. White Paper, Financial Services, Infosys Consulting.

[2012] Raiko T., Valpola H., LeCun Y., Deep learning made easier by linear transformations in perceptrons. In *International Conference on Artificial Intelligence and Statistics*, pp 924–932.

[2018] Raissi M., Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. Working Paper, arXiv:1804.07010.

[2016] Rajpurkar P., Zhang J., Lopyrev K., Liang P., Squad: 100,000+ questions for machine comprehension of text. CoRR, 2016.

[2009] Rakhlin A., Lecture notes on online learning. Draft.

[2016] Ramachandran P., Liu P.J., Le Q.V., Unsupervised pretraining for sequence to sequence learning. Working Paper, arXiv:1611.02683.

[1999] Ramsey J.B., The contribution of wavelets to the analysis of economic and financial data. *Phil. Trans. R. Soc.*, **357**, pp 2593–2606.

[2018] Ranganathana G., Bindhu V., Raj J., Duplicate record detection using intelligent approaches. *International Journal of Pure and Applied Mathematics*, **119**, (12), pp 13077–13087.

[2017] Rani T., Kumar N., A survey: Hybrid intelligent modeling technique for data classification. *International Journal of Advanced Research in Computer and Communication Engineering*, **6**, (5), pp 616–623.

[2014] Ranzato M., Szlam A., Bruna J., Mathieu M., Collobert R., Chopra S., Video (language) modeling: A baseline for generative models of natural videos. arXiv preprint arXiv:1412.6604.

[2000] Ratle A., Sebag M., Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. In Marc Schoenauer, Kalyanmoy Deb, Gunter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature*, PPSN 6th International Conference, volume 1917 of LNCS, pp 211–220, Springer Verlag.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2001] Ratsch G., Onoda T., Muller K-R., Soft margins for AdaBoost. *Machine Learning*, **42**, (3), pp 287–320.

[2004] Ravikumar P., Cohen W.W., A hierarchical graphical model for record linkage. In *20th Conf. Uncertainty in Artificial Intelligence (UAI '04)*.

[1987] Rebane G., Pearl J., The recovery of causal poly-trees from statistical data. In *Proceedings, 3rd Workshop on Uncertainty in AI*, Seattle, WA., pp 222–228.

[2013] Recchia G., Louwerse M.M., A comparison of string similarity measures for toponym matching. In *Proceedings of the International Workshop on Computational Models of Place SIGSPACIAL*.

[2015] Redmon J., Divvala S., Girshick R., Farhadi A., You only look once: Unified, real-time object detection. University of Washington.

[2018] Redmon J., Farhadi A., Yolov3: An incremental improvement. University of Washington.

[1995] Refenes A.N., Neural networks in capital markets. Wiley.

[1994] Refenes A.N., Zapranis A.D., Francis G., Stock performance modeling using neural network: A comparative study with regression models. *Neural Network*, **5**, pp 961–970.

[1997] Refenes A.N., Bentz Y., Bunn D.W., Burgess A.N., Zapranis A.D., Financial time series modeling with discounted least squares backpropagation. *Neurocomputing*, **14**, (2), pp 123–138.

[2017] Rei M., Semi-supervised multitask learning for sequence labeling. Working Paper, arXiv:1704.07156v1.

[2019] Reimers N., Gurevych I., Sentence-BERT: Sentence embeddings using siamese BERT-Networks. arXiv:1908.10084v1.

[2008] Reinhart R.F., Steil J.J., Recurrent neural associative learning of forward and inverse kinematics for movement generation of the redundant PA-10 robot. In Learning and Adaptive Behaviors for Robotic Systems (LAB-RS), pp 35–40.

[2009] Reinhart R.F., Steil J.J., Attractor-based computation with reservoirs for online learning of inverse kinematics. Research Institute for Cognition and Robotics (CoR-Lab), Bielefeld University.

[2009b] Reinhart R.F., Steil J.J., Reaching movement generation with a recurrent neural network based on learning inverse kinematics for the humanoid robot icub. In IEEE-RAS International Conference on Humanoid Robots (Humanoids), pp 323–330.

[2011] Reinhart R.F., Reservoir computing with output feedback. Technische Fakultät, Universität Bielefeld.

[2011] Reinhart R.F., Steil J.J., A constrained regularization approach for input-driven recurrent neural networks. *Differential Equations and Dynamical Systems*, **19**, pp 27–46.

[2004] Reisinger J., Stanley K.O., Miikkulainen R., Evolving reusable neural modules. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund Burke, Paul Darwen, Dipankar Dasgupta, Dario Floreano, James Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andy Tyrrell, editors, *Genetic and Evolutionary Computation*, GECCO-2004, Part II, volume 3103 of Lecture Notes in Computer Science, pp 69–81, Seattle, WA, USA. Springer-Verlag.

[1997] Reiss R., Thomas M., Statistical analysis of extreme values with applications to insurance, finance, hydrology and other fields. Birkhauser, Basel.

[2012] Rempis C.W., Evolving complex neuro-controllers with interactively constrained neuro-evolution. PhD Thesis, Universitat Osnabruck.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2017] Rendon de la Torre S., Kalda J., Kitt R., Engelbrecht J., Fractal and multifractal analysis of complex networks: Estonian network of payments. Working Paper, Tallinn University of Technology, Estonia,

[1967] Rescorla R.A., Pavlovian conditioning and its proper control procedures. *Psychological Review*, **74**, (1), pp 71–80.

[1999] Resnick S., Samorodnitsky G., Xue F., How misleading can sample ACFs of stable MAs be? (Very!). *Ann. Appl. Prob.*, **9**, pp 797–817.

[2000] Resnick S., Van Den Berg E., Sample correlation behavior for the heavy tailed general bilinear process. *Comm. Statist. Stochastic Models*, **16**, pp 233–258.

[1994] Reynolds C.W., Competition, coevolution and the game of tag. In Rodney A. Brooks and Pattie Maes, editors, *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pp 59–69. MIT Press.

[2014] Rezende D.J., Mohamed S., Wierstra D., Stochastic backpropagation and approximate inference in deep generative models. Technical Report, arXiv:1401.4082.

[2016] Ribeiro M.T., Singh S., Guestrin C., Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp 1135–1144.

[2018] Ribeiro J.M.L., Bravo P., Wang Y., Tiwary P., Reweighted autoencoded variational bayes for enhanced sampling (rave). *J. Chem. Phys.*, 149:072301.

[1995] Rice J.A., Mathematical statistics and data analysis. Thomson Information, Second Edition.

[2005] Riedmiller M., Neural fitted Q-iteration: Batch-mode Q-learning with neural networks. In J. Gama et al. (Eds.): ECML 2005, LNAI 3720, pp 317–328.

[2011] Rifai S., Vincent P., Muller X., Glorot X., Bengio Y., Contractive autoencoders: Explicit invariance during feature extraction. In *ICML*, pp 833–840.

[2009] Risi S., Vanderbleek S.D., Hughes C.E., Stanley K.O., How novelty search escapes the deceptive trap of learning to learn. In *Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation*, pp 153–160, ACM.

[2012] Risi S., Stanley K.O., An enhanced hypercube-based encoding for evolving the placement, density and connectivity of neurons. *Artificial Life journal* Cambridge, Massachusetts, MIT Press.

[1998] Ristad E.S., Yianilos P.N., Learning string edit distance. *IEEE Trans. Pattern Analysis and Machine Intelligence*, **20**, (5), pp 522–532.

[2011] Ritter A., Clark S., Etzioni M., Etzioni O., Named entity recognition in tweets: An experimental study. In *Proc. Empirical Methods in Natural Language Processing*.

[1990] Rivest R.L., Schapire R.E., A new approach to unsupervised learning in deterministic environments. *Machine learning: An artificial intelligence approach*, volume III, pp 670–684.

[1951] Robbins H., Monro S., A stochastic approximation method. *Ann. Math. Statistics*, **22**, pp 400–407.

[2007] Robert C.P., The Bayesian choice: From decision-theoretic foundations to computational implementation. New York, Springer, 2nd ed.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2009] Robert C.P., Wraith D., Computational methods for Bayesian model choice. In *AIP Conf. Proc.*, 1193, pp 251–62.

[1987] Robinson A.J., Fallside F., The utility driven dynamic error propagation network. Cambridge University Engineering Department

[1956] Rochester N., Holland J.H., Habit L.H., Duda W.L., Tests on a cell assembly theory of the action of the brain, using a large digital computer. *IRE Transactions on Information Theory*, **2**, (3), pp 80–93.

[1995] Rockafellar R.T., Convex analysis. Princeton University Press.

[2000] Rockafellar R.T., Uryasev S., Optimization of conditional value-at-risk. *Journal of Risk*, **2**, pp 21–41.

[2012] Rodan A., Tiňo P., Simple deterministically constructed cycle reservoirs with regular jumps. The University of Birmingham.

[1967] Rogers Jr H., Theory of recursive functions and effective computability. MIT Press (1987), Reprint of the 1967 edition, Cambridge MA.

[2010] Rogers L.C.G., Tehranchi M.R., Can the implied volatility surface move by parallel shifts? *Finance and Stochastics*, **14**, (2), pp 235–248.

[2019] Roh Y., Heo G., Whang S.E., A survey on data collection for machine learning: a big data - AI integration perspective. *IEEE TKDE*.

[2010] Rokach L., Pattern classification using ensemble methods. World Scientific, Singapore.

[1994] Ronald E., Schoenauer M., Genetic lander: An experiment in accurate neuro-genetic control. PPSN III, Parallel Programming Solving from Nature.

[2015] Ronneberger O., Fischer P., Brox T., U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, Springer, pp 234–241.

[2010] Roper M., Arbitrage free implied volatility surfaces. Working Paper, School of Mathematics and Statistics, The University of Sydney, Australia.

[1957] Rosenblatt F., The Perceptron: A perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

[1958] Rosenblatt F., The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, **65**, (6), pp 386–408.

[1962] Rosenblatt F., Principles of neurodynamics: Perceptrons and the theory of brain mechanics. Spartan, Washington, DC.

[1996] Ross S.M., Stochastic processes. Second Edition, John Wiley & Sons, Inc. First Edition 1983.

[1999] Ross B.J., A lamarckian evolution strategy for genetic algorithms. In Lance D. Chambers, editor, *Practical Handbook of Genetic Algorithms: Complex Coding Systems*, **3**, pp 1–16. CRC Press, Boca Raton, Florida.

[2017] Ross A.S., Hughes M.C., Doshi-Velez F., Right for the right reasons: Training differentiable models by constraining their explanations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, IJCAI-17, pp 2662–2670.

[2017] Rostami S., Neri N., A fast hypervolume driven selection mechanism for many-objective optimisation problems. *Swarm and Evolutionary Computation*, **34**, (Supplement C), pp 50–67.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2008] Rozenfeld H.D., Gallos L.K., Song C., Makse H.A., Fractal and transfractal scale-free networks. *Encyclopedia of Complexity and Systems Science*.

[1987] Rubin D., Multiple imputation for nonresponse in surveys. Wiley Series in Probability and Statistics. Wiley.

[1981] Rubinstein R.Y., Simulation and the Monte Carlo method. John Wiley & Sons, New York.

[2002] Rubinstein M., Markowitz's portfolio selection: A fifty-year retrospective. *The Journal of Finance*, **57**, (3), pp 1041–1045.

[2018] Ruder S., Bingel J., Augenstein I., Sogaard A., Latent multi-task architecture learning. Working Paper, arXiv:1705.08142v3.

[2019] Ruder S., Neural transfer learning for natural language processing. PhD in the College of Engineering and Informatics, National University of Ireland, GALWAY.

[1962] Rudin W., Fourier analysis on groups. Interscience Tracts in Pure and Applied Mathematics, No. 12, Interscience Publishers (a division of John Wiley and Sons), New York, London.

[1985] Rumelhart D.E., Hinton G.E., Williams R.J., Learning internal representations by error propagation. Technical Report No 8506, La Jolla: University of California, San Diego.

[1986b] Rumelhart D., McClelland J., Learning internal representations by error propagation. In *Parallel distributed processing: explorations in the microstructure of cognition*, **1**, MIT Press, Cambridge, Mass, pp 318–362.

[1986c] Rumelhart D.E., Hinton G.E., Williams R.J., Learning representations by back-propagating errors. *Nature*, **323**, pp 533–536.

[1994] Rumelhart D., Widrow B., Lehr M., The basic ideas in neural networks *Communications of the ACM*, **37**, (3), pp 87–92.

[1994] Rummery G.A., Niranjan M., Online Q-Learning using connectionist systems. Working Paper, Cambridge University Engineering Department.

[2014] Russakovsky O., Deng J., Su H., Krause J., Satheesh S., Ma S., Huang Z., Karpathy A., Khosla A., Bernstein M., Berg A.C., Fei-Fei L., ImageNet large scale visual recognition challenge. arXiv:1409.0575

[2009] Russell S.J., Norvig P., Artificial intelligence: A modern approach. Prentice Hall, (3rd ed.), (original version 1995, Prentice hall Englewood Cliffs), Upper Saddle River, New Jersey.

[2003] Rutherford A., Mathematical modelling techniques. Dover Publications Inc.; New edition edition. First published in 1978.

[1998] Ryan R., Collins J.J., O'Neill M., Grammatical evolution: evolving programs for an arbitrary language. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of LNCS, pp 83–95, Springer-Verlag.

[2018] Sabokrou M., Khalooei M., Fathy M., Adeli E., Adversarially learned one-class classifier for novelty detection. In *CVPR*, pp 3379–3388.

[2011] Sadiq S., Yeganeh N.K., Indulska M., 20 years of data quality research: Themes, trends and synergies. In *Proceedings of the Twenty-Second Australasian Database Conference*, **115**, Darlinghurst, Australia, Australian Computer Society, Inc., pp 153–162.

[2009] Salakhutdinov R., Hinton G.E., Deep Boltzmann machine. In *AISTATS 2009*, pp 448–455.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2013] Salimans T., Knowles D.A., Fixed-form variational posterior approximation through stochastic linear regression. *Bayesian Analysis*, **8**, (4).

[1983] Salton G., McGill M., Introduction to modern information retrieval. McGraw-Hill.

[2018] Samsuddin A.F., Retnawati H., Mathematical representation: The roles, challenges and implication on instruction. *Journal of Physics Conference Series*, 1097 012152.

[1959] Samuel A.L., Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, **3**, (3), pp 535–554. Reprinted in E.A. Feigenbaum & J. Feldman (Eds.), Computers and thought. New York, McGraw-Hill.

[1983] Samuel A.L., AI: Where it has been and where it is going. In Proceedings of the Eighth International Joint Conference on Artificial Intelligence, pp 1152–1157, Los Altos, CA. Morgan Kaufman.

[1965] Samuelson P.A., Proof that properly anticipated prices fluctuate randomly. *Industrial Management Review*, **6**, pp 41–50.

[1969] Samuelson P.A., Merton R.C., A complete model of warrant pricing that maximizes utility. *Industrial Management Review*, pp 17–46.

[2015] Sangwan K.S., Saxena S., Kant G., Optimization of machining parameters to minimize surface roughness using integrated ANN-GA approach. In *Proceedings of the 22nd CIRP Conference on Life Cycle Engineering*, **29**, Sydney, Australia, pp 305–310.

[2005] Santana-Quintero L.V., Coello Coello C.A., An algorithm based on differential evolution for multi-objective problems. *International Journal of Computational Intelligence Research*, **1**, ISSN 0973-1873, pp 151–169.

[2017] Santos R., Murrieta-Flores P., Martins B., Learning to combine multiple string similarity metrics for effective toponym matching. *International Journal of Digital Earth*, **11**, (9).

[2018] Santos R., Murrieta-Flores P., Calado P., Martins B., Toponym matching through Deep Neural Networks. *International Journal of Geographical Information Science*, **32**, (2).

[1995] Saravanan N., Fogel D.B., Evolving neural control systems. *IEEE Expert*, pp 23–27.

[2002] Sarawagi S., Bhamidipaty A., Interactive deduplication using active learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, Edmonton, Alberta.

[1998] Sareni B., Krahenbuhl L., Fitness sharing and niching methods revisited. *Evolutionary Computation*, IEEE Transactions on, **2**, (3), pp 97–106.

[1994] Sarle W.S., Neural networks and statistical models. In *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, April, 1994.

[1954] Savage L.J., The foundations of statistics. Wiley, New York, second revised edition: Dover, New York, 1972.

[1984] Schaffer J.D., Some experiments in machine learning using vector evaluated genetic algorithms. PhD dissertation, Vanderbilt University, Nashville, TN.

[1992] Schaffer J.D., Whitley D., Eshelman L.J., Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *Proceedings of COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks*.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1982] Schantz H.F., The History of OCR, optical character recognition. Recognition Technologies Users Association.

[1990] Schapire R.E., The strength of weak learnability. *Machine Learning*, **5**, (2), pp 197–227.

[2012] Schaul T., Zhang S., LeCun Y., No more pesky learning rates. arXiv preprint arXiv:1206.1106.

[2016] Schaul T., Quan J., Antonoglou I., Silver D., Prioritized experience replay. In *International Conference on Learning Representations* ICLR.

[2010] Scherer D., Muller A., Behnke S., Evaluation of pooling operations in convolutional architectures for object recognition. In *International Conference on Artificial Neural Networks*, pp 92–101.

[2017] Schlegl T., Seebock P., Waldstein S.M., Schmidt-Erfurth U., Langs G., Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *IPMI*, Springer, Cham, pp 146–157.

[2019] Schlegl T., Seebock P., Waldstein S.M., Langs G., Schmidt-Erfurth U., f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks. *Medical Image Analysis*, **54**, pp 30–44.

[1992] Schmidhuber J., A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running network. *Neural Computation*, **4**, (2), pp 243–248.

[1992b] Schmidhuber J., Learning factorial codes by predictability minimization. *Neural Computation*, **4**, (6), pp 863–879.

[2007] Schmidhuber J., Wierstra D., Gagliolo M., Gomez F.J., Training recurrent networks by Evolino. *Neural Computation*, **19**, (3), pp 757–779.

[1999] Schmitt F., Schertzer D., Lovejoy S., Multifractal analysis of foreign exchange data. *Applied Stochastic Models and Data Analysis*, **15**, pp 29–53.

[1996] Schoenauer M., Sebag M., Jouve F., Lamy B., Maitournam H., Evolutionary identification of macro-mechanical models. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 23, pp 467–488. MIT Press, Cambridge, MA, USA.

[2015] Schulman J., Moritz P., Levine S., Jordan J., Abbeel P., Trust region policy optimisation: deep RL with natural policy gradient and adaptive step size. University of California, Berkeley, Department of Electrical Engineering and Computer Sciences.

[2016] Schulman J., Moritz P., Levine S., Jordan J., Abbeel P., High-dimensional continuous control using generalized advantage estimation: Batch-mode actor-critic with blended Monte Carlo and function approximator returns. Published as a conference paper at ICLR 2016

[2005] Schulmeister S., Components of the profitability of technical currency trading. WIFO Working Papers, No. 263, December.

[1969] Schumacker R.A., Brand B., Gilliland M.G., Sharp W.H., Study for applying computer-generated images to visual simulation. U.S. Air Force Human Resources Laboratory, pp 142.

[1997] Schuster M., Paliwal K.K., Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, **45**, (11).

[1996] Schwager W.F., Technical analysis. Wiley.

[1995] Schwefel H.P., Evolution and optimum seeking. John Wiley & Sons.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1988] Schweizer M., Hedging of options in a general semimartingale model. Diss. ETHZ, no 8615, Zurich.

[1995] Schweizer M., Variance-optimal hedging in discrete time. *Mathematics of Operations Research*, **20**, pp 1–32.

[2012] Schwenk H., Continuous space translation models for phrase-based statistical machine translation. In M. Kay and C. Boitet editors, *Proceedings of the 24th International Conference on Computational Linguistics (COLIN)*, pp 1071–1080.

[2008] Senol D., Ozturan M., Stock price direction prediction using artificial neural network approach: The case of Turkey. *Journal Artif Intell*, **1**, (2), pp 70–77.

[2006] Sessions V., Valtorta M., The effects of data quality on machine learning algorithms. In *Proceedings of the 11th International Conference on Information Quality*, MIT, Cambridge, MA, USA, November 10-12.

[2015] Severyn A., Moschitti A., Unitn: Training deep convolutional neural network for twitter sentiment classification. In *Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015)*, pp 464–469.

[2017] Shabanian S., Arpit D., Trischler A., Bengio Y., Variational Bi-LSTMs. arXiv preprint arXiv:1711.05717.

[2011] Shalev-Shwartz S., Singer Y., Srebro N., Cotter A., Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming*, **127**, (1), pp 3–30.

[1938] Shannon C.E., A symbolic analysis of relay and switching circuits. *Trans. AIEE.*, **57**, (12), pp 713–723.

[1948] Shannon C.E., A mathematical theory of communication. *Bell System Technical Journal*, **27**, (3), pp 379–423.

[1949] Shannon C.E., Communication in the presence of noise. *Proc. I.R.E.*, **37**, pp 10–21.

[1949b] Shannon C.E., The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, **28**, pp 59–98.

[1949] Shannon C.E., Weaver W., The mathematical theory of communication. University of Illinois Press, Urbana, Illinois.

[2016] Shao X., Zhang Z., Wang J., Zhang D., NSGA-II-based multi-objective mission planning method for satellite formation system. *J. Aerosp. Technol. Manag.*, **8**, (4).

[2017] Shao Y., Use convolutional neural network to evaluate semantic textual similarity. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp 130–133.

[2015] Shaoqing R., Faster R-CNN. *Advances in Neural Information Processing Systems*. arXiv:1506.01497.

[1981] Shapiro E.Y., The model inference system. In *Proceedings of the 7th international joint conference on Artificial intelligence-Volume 2*, Morgan Kaufmann Publishers Inc.

[1983] Shapiro E.Y., Algorithmic program debugging. Cambridge, Mass, MIT Press.

[1964] Sharpe W.F., Capital asset prices: A theory of market equilibrium under conditions of risk. *Journal of Finance*, **19**, (3), pp 425–442.

[2016] Shen F., Gan R., Zeng G., Weighted residuals for very deep networks. In *Proc. 3rd Int. Conf. Syst. Informat. (ICSAI)*, pp 936–941.

[2008] Sherrod P.H., Dtreg: predictive modeling software.

[2015] Shi X., Chen Z., Wang H., Yeung D-Y., Wong W-K., Woo W-C., Convolutional LSTM network: A machine learning approach for precipitation nowcasting. arXiv:1506.04214v2.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1991] Shin Y., Ghosh J., The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. *International Joint Conference on Neural Networks*.

[1998] Shin K-S., Kim K-J., Han I., Financial data mining using genetic algorithms technique: Application to KOSPI 200. Korea Advanced Institute of Science and Technology

[2000] Shin T., Han I., Optimal signal multi-resolution by genetic algorithms to support financial neural networks for exchange-rate forecasting. *Expert Syst. Appl.*, **18**, pp 257–269.

[2002] Shipp C.A., Kuncheva L.I., Relationships between combination methods and measures of diversity in combining classifiers. *Information Fusion*, **3**, (2), pp 135–148.

[1997] Shleifer A., Vishny R.W., The limits of arbitrage. *Journal of Finance*, **52**, (1), pp 35–55.

[1985] Shor N.Z., Minimization methods for non-differentiable functions. Springer-Verlag.

[2003] Shreve S.E., Stochastic calculus for finance I. The binomial asset pricing model. Springer Finance, Textbook, NY, USA.

[2004] Shreve S.E., Stochastic calculus for finance II, Continuous-time models. Springer Finance, Textbook, NY, USA.

[2012] Shynkevich A., Performance of technical analysis in growth and small cap segments of the US equity market. *Journal of Banking & Finance*, **36**, pp 193–208.

[2012] Si T., Hazra S., Jana N.D., Artificial neural network training using differential evolutionary algorithm for classification. Department of Information Technology National Institute of Technology, Durgapur West Bengal, India.

[1991] Siegelmann H.T., Sontag E.D., Turing computability with neural nets. *Applied Mathematics Letters*, **4**, (6), pp 77–80.

[2007] Siewert U., Wustlich W., Echo-state networks with bandpass neurons: Towards generic time-scale-independent reservoir structures. Internal Status Report, *PLANET Intelligent Systems GmbH*.

[1996] Silberschatz A., Korth H.F., Sudarshan S., Data models. *ACM Comput. Surv.*, **28**, (1), pp 105–108.

[2010] Silberschatz A., Database system concepts. Sixth Edition, McGraw-Hill.

[2012] Silver D., Gradient temporal difference networks, *JMLR, Workshop and Conference Proceedings*, **24**, pp 117–129, 2012 10th European Workshop on Reinforcement Learning.

[2014] Silver D., Lever G., Heess N., Degris T., Wierstra D., Riedmiller M., Deterministic policy gradient algorithms. In *International Conference on Machine Learning (ICML)*, 2014.

[1955] Simon H.A., On a Class of Skew Distribution Functions. *Biometrika*, **42**, (3/4), pp 425–440.

[2013] Simon D., Evolutionary optimization algorithms: Biologically inspired and population-based approaches to computer intelligence. Hoboken, Wiley.

[2018] Simonini T., An introduction to reinforcement learning. Deep reinforcement learning course.

[2015] Simonyan K., Zisserman A., Very deep convolutional networks for large-scale image recognition. In ICLR.

[2004] Singla P., Domingos P., Multi-relational record linkage. In *Proc. KDD-2004 Workshop Multi-Relational Data Mining*, pp 31–48.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2005] Singla P., Domingos P., Object identification with attribute mediated dependences. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PAKDD-2005)*, Porto, Portugal.

[1996] Skalak D.B., The sources of increased accuracy for two proposed boosting algorithms. In *Working Notes of the AAAI'96 Workshop on Integrating Multiple Learned Models*, Portland, OR.

[2008] Slowik A., Bialko M., Training of artificial neural networks using differential evolution algorithm. Department of Electronics and Computer Science, Koszalin University of Technology, Koszalin, Poland.

[2004] Smart W., Zhang M., Applying online gradient descent search to genetic programming for object recognition. In James Hogan, Paul Montague, Martin Purvis, and Chris Steketee, editors, CRPIT 04: Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation, volume 32, no. 7, pp 133–138, Dunedin, New Zealand. Australian Computer Society.

[1989] Smith M.J., Portmann C.L., Practical design and analysis of a simple neural optimization circuit. *IEEE Trans. Circuit and Systems*, **36**, pp 42–50.

[2020] Smith N.A., Contextual word representations: A contextual introduction. Working Paper, arXiv:1902.06006v3.

[1973] Sneath P.H.A., Sokal R.R., Numerical taxonomy: The principles and practice of numerical classification. W.H. Freeman, San Francisco, CA.

[1982] Sobel M., The variance of discounted Markov decision processes. *Journal of Applied Probability*, **19**, pp 794–802.

[2013] Socher R., Perelygin A., Wu J., Chuang J., Manning C.D., Ng A., Potts C., Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, pp 1631–1642.

[2017] Soltani A.A., Huang H., Wu J., Kulkarni T.D., Tenenbaum J.B., Synthesizing 3D shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp 1511–1519.

[2017] Sompolinsky H., Lectures on reinforcement learning. Harvard Canvas.

[2015] Sondwale P., Overview of predictive and descriptive data mining Techniques. *International Journal of Advanced Research in Computer Science and Software*, **5**, (4), pp. 262–265.

[2005] Song C., Havlin S., Makse H.A., Self-similarity of complex networks. *Nature*, **433**, pp 392–395.

[2019] Song K., Tan X., Qin T., Lu J., Liu T-Y., MASS: Masked sequence to sequence pre-training for language generation. arXiv:1905.02450.

[2010] Sorensen K., Springael J., Busschaert S., Sequential versus simultaneous application of multi-objective optimization and multi-criteria decision making: An empirical investigation. Working Papers 2010023, University of Antwerp, Faculty of Business and Economics.

[2008] Sorokina D., Caruana R., Riedewald M., Fink D., Detecting statistical interactions with additive groves of trees. In *Proceedings of the 25th international conference on Machine learning*, pp 1000–1007

[1994] Spencer G.F., Automatic generation of programs for crawling and walking. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 15, pp 335–353. MIT Press.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1961] Sprenkle C., Warrant prices as indications of expectations. *Yale Econ. Essays 1*, pp 179–232. Reprinted in Cootner (1967), pp 412–474.

[2019] Srikant R., Ying L., Finite-time error bounds for linear stochastic approximation and TD learning. *Conference on Learning Theory*, 2019.

[2014] Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R., Dropout: A simple way to prevent neural networks from over

tting. *The Journal of Machine Learning Research*, **15**, (1), pp 1929–1958.

[2015] Srivastava R.K., Greff K., Schmidhuber J., Highway networks. arXiv:1505.00387.

[2015b] Srivastava R.K., Greff K., Schmidhuber J., Training very deep networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.

[2015c] Srivastava N., Mansimov E., Salakhutdinov R., Unsupervised learning of video representations using lstms. In *ICML*.

[2000] Stanley H.E., Amaral L.A.N., Scala A., Barthelemy M., Classes of small-world networks. *PNAS*, **97**, (21), pp 11149–11152.

[2002a] Stanley K.O., Miikkulainen R., Evolving neural network through augmenting topologies. *Evolutionary Computation*, **10**, (2), pp 99–127.

[2002b] Stanley K.O., Miikkulainen R., Efficient reinforcement learning through evolving neural network topologies. In William B. Langdon, Erick Cantu-Paz, Keith E. Mathias, Rajkumar Roy, David Davis, Riccardo Poli, Karthik Balakrishnan, Vasant Honavar, Gunter Rudolph, Joachim Wegener, Larry Bull, Mitchell A. Potter, Alan C. Schultz, Julian F. Miller, Edmund K. Burke, and Natasa Jonoska, editors, GECCO, pp 569–577. Morgan Kaufmann.

[2003] Stanley K.O., Miikkulainen R., A taxonomy for artificial embryogeny. *Artificial Life*, **9**, (2), pp 93–130.

[2004] Stanley K.O., Efficient evolution of neural networks through complexification. PhD thesis, The University of Texas at Austin.

[2007] Stanley K.O., Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, **8**, (2), pp 131–162.

[2009] Stanley K.O., D'Ambrosio D.B., Gauci J., A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, **15**, (2), pp 185–212.

[2017] Stark L., Machine learning and options pricing: A comparison of Black-Scholes and a deep neural network in pricing and hedging DAX 30 index options. Department of Finance, Aalto University School of Business.

[2004] Steil J.J., Backpropagation-decorrelation: Online recurrent learning with O(N) complexity. Neuroinformatics Group, Faculty of Technology University of Bielefeld. In *Proceedings of the IEEE International Joint Conference on Artificial Neural Networks*, **2**, pp 843–848.

[2005] Steil J.J., Memory in backpropagation-decorrelation $O(N)$ efficient online recurrent learning. In *Proceedings of the 15th International Conference on Artificial Neural Networks*, **3697** of LNCS, pp 649–654.

[2005] Steinkraus D., Simard P., Buck I., Using GPUs for machine learning algorithms. In *12th International Conference on Document Analysis and Recognition*, pp 1115–1119.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2012] Stepanek J., Stovicek J., Cimler R., Application of genetic algorithms in stock market simulation. Cyprus International Conference on Educational Research (CY-ICER-2012) North Cyprus, US08-10.

[1999] Stephens C.R., Waelbroeck H., Schemata evolution and building blocks. *Evolutionary Computation*, **7**, (2), pp 109–124.

[2006] Steyvers M., Griffiths T., Probabilistic topic models. In T. Landauer, D. McNamara, S. Dennis, and W. Kintsch eds, *Latent Semantic Analysis: A Road to Meaning*, Laurence Erlbaum.

[1972] Stone H.S., Introduction to computer organization and data structures. McGraw-Hill, New York.

[2018] Stone H., Calibrating rough volatility models: A convolutional neural network approach. Preprint, arXiv:1812.05315.

[2019] Stonebraker M., Rezig E.K., Machine learning and big data: What is important? *IEEE Data Eng. Bull.*.

[1995] Storn R., Price K., Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. International Computer Science Institute, Berkeley, **TR-95-012**.

[1996] Storn R., System design by constraint adaptation and differential evolution. International Computer Science Institute, Berkeley, **TR-96-039**.

[1997] Storn R., Price K., Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, **11**, (4), pp 341–359.

[2000] Storn R., Digital filter design program. FIWIZ.

[2008] Storn R., Differential evolution research: Trends and open questions. in U.K. Chakraborty, editor, *Advances in Differential Evolution*, **1**, pp 1–31, Springer-Verlag, Berlin Heidelberg.

[2014] Stosic D., Stosic D., Stosic T., Stanley H.E., Multifractal analysis of managed and independent float exchange rates. Department of Physics, Boston University. Published in 2015 in *Physica A: Statistical Mechanics and its Applications*, **428**, pp 13–18.

[1994] Strichartz R., A guide to distribution theory and Fourier transforms. CRC Press.

[1994] Strogatz S.H., Nonlinear dynamics and chaos: With applications to physics, biology chemistry and engineering. Addison Wesley.

[1998] Struzik Z.R., Removing divergence in the negative moments of the multi-fractal partition function with the wavelet transformation. Working Paper INS-R9803, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

[1999] Struzik Z.R., Local effective Holder exponent estimation on the wavelet transform maxima tree. in *Fractals: Theory and Applications in Engineering*, eds., M. Dekking, J. Levy Vehel, E. Lutton, C. Tricot, Springer Verlag, pp 93–112.

[2000] Struzik Z.R., Determining local singularity strengths and their spectra with the wavelet transform. *Fractals*, **8**, (2), pp 163–179.

[2002] Struzik Z.R., Siebes A., Wavelet transform based multifractal formalism in outlier detection and localisation for financial time series. *Physica A*, **309**, pp 388–402.

[2003] Struzik Z.R., Econophysics vs cardiophysics: The dual face of multifractality. Working Paper, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2014] Suhov Y.M., Kelbert M., Probability and statistics by example. Second edition, Cambridge University Press.

[1999] Sun X., The Lasso and its implementation for neural networks. Department of Statistics, University of Toronto.

[1999] Sun W., Yuan Y-X., Optimization theory and methods: Nonlinear programming. Springer.

[2010] Sundermeyer M., Schluter R., Ney H., LSTM neural networks for language modeling. In *INTERSPEECH*.

[2009] Sussillo D., Abbott L.F., Generating coherent patterns of activity from chaotic neural networks. *Neuron*, **63**, (4), pp 544–557.

[1995] Sutinen E., Tarhio J., On using q-gram locations in approximate string matching. *Proc. Third Ann. European Symp. Algorithms (ESA '95)*, pp 327–340.

[2008] Sutskever I., Hinton G., Taylor G.W., The recurrent temporal restricted boltzmann machine. In *Advances in Neural Information Processing Systems 21*, (NIPS 2008), pp 1601–1608.

[2011] Sutskever I., Martens J., Hinton G., Generating text with recurrent neural networks. In *ICML*, 2011.

[2012] Sutskever I., Training recurrent neural networks. PhD thesis, University of Toronto.

[2013] Sutskever I., Martens J., Dahl G., Hinton G., On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*.

[2014] Sutskever I., Vinyals O., Le Q.V., Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp 3104–3112.

[1981] Sutton R.S., Barto A.G., Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review*, **88**, pp 135–171.

[1984] Sutton R.S., Temporal credit assignment in reinforcement learning. Ph.D. thesis, University of Massachusetts, Amherst.

[1988] Sutton R.S., Learning to predict by the methods of temporal differences. *Machine Learning*, **3**, pp 9–44.

[1989] Sutton R.S., Implementation details of the $TD(\lambda)$ procedure for the case of vector predictions and back-propagation. GTE Laboratories Incorporated., Technical Note TN87-509.1.

[2000] Sutton R.S., McAllester D.A., Singh S.P., Mansour Y., Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, **99**, pp 1057-1063.

[2009] Sutton R.S., Maei H.R., Precup D., Bhatnagar S., Silver D., Szepesvari C., Wiewiora E., Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *26th Annual International Conference on Machine Learning*, pp 125.

[2017] Sutton R.S., Barto A.G., Reinforcement learning: An introduction. Second Edition, MIT Press, Cambridge, MA. First edition is from 1998.

[1986] Sweeney R., Beating the foreign exchange market. *The Journal of Finance*, **41**, (1), pp 163–182.

[2010] Szeliski R., Computer vision: Algorithms and Applications. Springer Science & Business Media.

[2010] Szepesvari C., Algorithms for reinforcement learning. Morgan & Claypool Publishers.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1996] Takefuji Y., Wang J., Neural computing for optimization and combinatorics. World Scientific Publishing Co.

[2013] Talburt J.R., Special issue on entity resolution overview: The criticality of entity resolution in data and information quality. *J. Data and Information Quality*, **4**, (2), pp 6:1–6:2.

[2018] Talman A., Yli-Jyra A., Tiedemann J., Natural language inference with hierarchical BiLSTM max pooling architecture. arXiv preprint arXiv:1808.08762.

[1995] Tan H., Neural network model for stock forecasting. Master's Thesis, Texas Tech University.

[2009] Tan C., Financial time series forecasting using improved wavelet neural network. Master's Thesis, Aarhus Universitet.

[2007] Tanaka-Yamawaki M., Tokuoka S., Adaptive use of technical indicators for the prediction of intra-day stock prices. *Physica A*, **383**, pp 125–133.

[2014] Tang F., Tiňo P., Chen H., Learning the deterministically constructed echo state networks. International Joint Conference on Neural Networks.

[1986] Tank D.W., Hopfield J.J., Simple neural optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit. *IEEE Trans. Circuits and Systems*, CAS-33, pp 533–541.

[2010] Tankov P., Touzi N., Calcul stochastique en finance. Ecole Polytechnique Paris, Departement de Mathematiques Appliquees.

[2005a] Tanner B., Sutton R.S., Temporal-difference networks with history. In *19th International Joint Conference on Artificial Intelligence*, pp 865–870.

[2005b] Tanner B., Sutton R.S., Td(lambda) networks: Temporal-difference networks with eligibility traces. In *22nd International Conference on Machine Learning*, pp 888–895.

[1981] Taqqu M.S., Self-similar processes and related ultraviolet and infrared catastrophes. In Random Fields : Rigorous Results in Statistical Mechanics and Quantum Field Theory, *Colloquia Mathematica Societatis Janos Bolya*, Vol. 27, Book 2, pp 1027–1096.

[1995] Taqqu M.S., Teverovsky V., Willinger W., Estimators for long-range dependence: An empirical study. *Fractals*, **3**, (4), pp 785–798.

[1942] Taylor S.J., Review: Introduction to logic and to the methodology of deductive sciences. In Curry, Haskell B. *Bull. Amer. Math. Soc.*, **48**, (7), pp 507–510.

[1953] Taylor W.L., Cloze procedure: A new tool for measuring readability. *Journalism Bulletin*, **30**, (4), pp 415–433.

[1986] Taylor S.J., Modelling financial time series. New York, John Wiley & Sons.

[2007] Taylor G.W., Hinton G.E., Roweis S., Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems*, **19**, pp 1345–1352, MIT Press, Cambridge.

[1867] Tchebichef P., Des valeurs moyennes. *Journal de mathematiques pures et appliquees*, **2**, (12), pp 177–184.

[2018] Tegmark M., Life 3.0. Penguin Books.

[2001] Tejada S., Knoblock C.A., Minton S., Learning object identification rules for information integration. *Information Systems Journal*, **26**, (8), pp 635–656.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2002] Tejada S., Knoblock C.A., Minton S., Learning domain independent string transformation weights for high accuracy object identification. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, Edmonton, Alberta.

[1989] Tel T., Fulop A., Vicsek A., Determination of fractal dimensions for geometrical multifractals. *Physica A: Statistical Mechanics and its Applications*, **159**, (2), pp 155–166.

[2016] Telgarsky M., Benefits of depth in neural networks. *JMLR: Workshop and Conference Proceedings 49*, pp 1–23.

[1994] Teller A., Genetic programming, indexed memory, the halting problem, and other curiosities. In *Proceedings of the 7th annual Florid Artificial Intelligence Research Symposium*, pp 270–274, Pensacola, Florida, USA. IEEE Press

[1997] Teller A., Andre D., Automatically choosing the number of fitness cases: The rational allocation of trials. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp 321–328, Stanford University, CA, USA. Morgan Kaufmann.

[1968] Tepping B.J., A model for optimum linkage of records. *J. Am. Statistical Assoc.*, **63**, (324), pp 1321–1332.

[2008] Terman D.H., Izhikevich E.M., State space. Scholarpedia, http://www.scholarpedia.org/article/Phase space, Ohio State University.

[1992] Tesauro G., Practical issues in temporal difference learning. *Machine Learning*, **8**, pp 257–277.

[2020] Thakur N., Reimers N., Daxenberger J., Gurevych I., Augmented SBERT: Data Augmentation Method for Improving Bi-encoders for pairwise sentence scoring tasks. Department of Computer Science, Technische Universitat Darmstadt, arXiv:2010.08240v1.

[1999] Theiler J.P., Harvey N.R., Brumby S.P., Szymanski J.J., Alferink S., Perkins S.J., Porter R.B., Bloch J.J., Evolving retrieval algorithms with a genetic programming scheme. In Michael R. Descour and Sylvia S. Shen, editors, *Proceedings of SPIE 3753 Imaging Spectrometry V*, pp 416–425.

[1967] Thorp E.O., Kassouf S.T., Beat the market. New York: Random House.

[2012] Tieleman T., Hinton G., Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Technical report.

[1963] Tikhonov A., Solution of incorrectly formulated problems and the regularization method. *Soviet Math. Dokl.*, **5**, pp 1035.

[2015] Tillmann A.M., On the computational intractability of exact and approximate dictionary learning. *IEEE Signal Processing Letters*, **22**, (1), pp 45–49.

[2004] Timmermann A., Granger C.W.J., Efficient market hypothesis and forecasting. *International Journal of Forecasting*, **20**, pp 15–27.

[2011] Tirri K., Nokelainen P., Measuring multiple intelligences and moral sensitivities in education. Sense Publishers, Series Volume 5.

[2003] Tjong Kim Sang E.F., De Meulder F., Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. CoNLL.

[1992] Todorovic P., An introduction to stochastic processes and their applications. Springer Series in Statistics, Springer-Verlag.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2017] Tolstikhin I.O., Gelly S., Bousquet O., Simon-Gabriel C-J., Scholkopf B., Adagan: Boosting generative models. in *Advances in Neural Information Processing Systems*, pp 5424–5433.

[2010] Touzi N., Optimal stochastic control, stochastic target problems, and backward SDE. Ecole Polytechnique, Departement de Mathematiques Appliquees.

[1965] Treynor J.L., How to rate management of investment funds. *Harvard Business Review*, **43**, (1), pp 63–75.

[1979] Trunk G.V., A problem of dimensionality: A simple example. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1 (3), pp 306–307.

[1939] Tryon R.C., Cluster analysis: Correlation profile and orthometric (factor) analysis for the isolation of unities in mind and personality. Edwards Brothers.

[2018] Tsang M., Cheng D., Liu Y., Detecting statistical interactions from neural network weights. Conference Paper at ICLR 2018, Department of Computer Science, University of Southern California.

[2002] Tsay R.S., Analysis of financial time series. John Wiley & Sons, Hoboken, New Jersey.

[1993] Tsitsiklis J., Bertsimas D., Simulated annealing. *Statistical Science*, **8**, (1), pp 10–15.

[1997] Tsitsiklis J.N., Van Roy B., An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, **42**, pp 674–690.

[1979] Tsukamoto Y., An approach to fuzzy reasoning method. In Mada M. Gupta, Rammohan K. Ragade and Roland R. Yager, editors, *Advances in Fuzzy Set Theory and Applications*, pp 137–149, North-Holland, Amsterdam.

[2010] Turian J., Ratinov L., Bengio Y., Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp 384–394. Association for Computational Linguistics.

[2006] Turiel A., Perez-Vicente C.J., Grazzini J., Numerical methods for the estimation of multifractal singularity spectra on sampled data: A comparative study. *Journal of Computational Physics*, **216**, pp 362–390.

[1936-37] Turing A., On computable numbers, with an application to the Entscheidungsproblem. in *Proceedings of the London Mathematical Society*, Series 2, (42), pp 230–265.

[1939] Turing A., Systems of logic based on ordinals. in *Proceedings of the London Mathematical Society*, (45), pp 161-228.

[1948] Turing A., Intelligent machinery. Report for National Physical Laboratory. Reprinted in Ince, D. C. (editor). 1992. Mechanical Intelligence: Collected Works of A. M. Turing. Amsterdam: North Holland. Pages 107127. Also reprinted in Meltzer, B. and Michie, D. (editors). 1969. Machine Intelligence 5. Edinburgh: Edinburgh University Press, 1948.

[1950] Turing A., Computing machinery and intelligence. *Mind*, **236**, pp 433–460.

[2010] Turney P.D., Pantel P., From frequency to meaning: Vector space models of semantics. *J. Artificial Intelligence Research*, **37**, (1), pp 141–188.

[1989] Uiterwijk J.W., van den Herik H.J., Allis L.V., A knowledge-based approach to connect-four. In David Levy and Don Beals, editors, *Heuristic Programming in Arti*

cial Intelligence: The First Computer Olympiad, Ellis Harwood, John Wiley.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1992] Ukkonen E., Approximate string matching with q-Grams and maximal matches. *Theoretical Computer Science*, **92**, (1), pp 191–211.

[2009] Valsalam V.K., Miikkulainen R., Evolving symmetric and modular neural networks for distributed control. In *Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation*, pp 731–738. ACM.

[2010] Valsalam V.K., Utilizing symmetry in evolutionary design. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, Austin, TX.

[2016] Van den Poel D., Chesterman C., Koppen M., Ballings M., Equity price direction prediction for day trading: Ensemble classification using technical analysis indicators with interaction effects. in *2016 IEEE Congress on Evolutionary Computation* (CEC).

[1997] Vandewalle N., Ausloos M., Coherent and random sequences in financial fluctuations. *Physica A*, **246**, (3), pp 454–459.

[1998] Vandewalle N., Ausloos M., Crossing of two mobile averages: A method for measuring the robustness exponent. *Phys. Rev.*, **58**, pp 177–188.

[1998b] Vandewalle N., Ausloos M., Multi-affine analysis of typical currency exchange rates. *European Physical Journal B.*, **4**, (2), pp 257–261.

[1998c] Vandewalle N., Ausloos M., Boveroux PH., Detrended fluctuation analysis of the foreign exchange market. Working Paper.

[2015] van Hasselt H., Guez A., Silver D., Deep reinforcement learning with double Q-learning. Google DeepMind.

[1963] Vapnik V., Lerner A., Pattern recognition using generalized portrait method. *Automation and Remote Control*, **24**, pp 774–780.

[1964] Vapnik V., Chervonenkis A., A note on one class of perceptrons. *Automation and Remote Control*, **25**.

[1982] Vapnik V., Estimation of dependencies based on empirical data. Springer-Verlag, New York.

[1989] Vapnik V., Chervonenkis A., Necessary and sufficient conditions for consistency of the method of empirical risk minimization. in Yearbook of the Academy of Sciences of the USSR on Recognition, Classification, and Forecasting, 2, Nauka (Moscow), pp 217–249.

[1992] Vapnik V., Principles of risk minimization for learning theory. AT&T Bell Laboratories.

[1995-2000] Vapnik V., The nature of statistical learning theory. Information Science and Statistics. Springer-Verlag. First published in 1995.

[2017] Vargas D.V., Murata J., Spectrum-diverse neuroevolution with unified neural models. IEEE Transactions on Neural Networks and Learning Systems, **28**, (8), pp 1759–1773.

[2017] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L., Polosukhin I., Attention is all you need. In *Advances in Neural Information Processing Systems*, pp 6000–6010.

[2016] Venna S., Gottumukkala1 R., Raghavan V., Visual analytic decision-making environments for large-scale time-evolving graphs. In *Cognitive Computing: Theory and Applications, ser. Handbook of Statistics*, V. Gudivada, V. Raghavan, V. Govindaraju, and C. R. Rao, Eds. New York, NY: Elsevier, **35**, pp 81–115.

[2018] Verissimo de Sousa Marinho A., Approximate string matching and duplicate detection in the deep learning era. MSc in Computer Engineering, Tecnico Lisboa.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2000] Verykios V.S., Elmagarmid A.K., Houstis E.N., Automating the approximate record matching process. *Information Sciences*, **126**, (1-4), pp 83–98.

[2003] Verykios V.S., Moustakides G.V., Elfeky M.G., A Bayesian decision model for cost optimal record matching. *VLDB J.*, **12**, (1), pp 28–40.

[2004] Verykios V.S., Moustakides G.V., A generalized cost optimal decision model for record matching. In *Proc. 2004 Int'l Workshop Information Quality in Information Systems*, pp 20–26.

[2017] Villarrubia G., De Paz J.F., Chamoso P., De la Prieta F., Artificial neural networks used in optimization problems. *Neurocomputing*, **272**, pp 10–16.

[2010] Vincent P., Larochelle H., Lajoie I., Bengio Y., Manzagol P-A., Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, December, pp 3371–3408.

[2015a] Vinyals O., Bengio S., Kudlur M., Order matters: Sequence to sequence for sets. In a conference paper at ICLR 2015.

[2015b] Vinyals O., Fortunato M., Jaitly N., Pointer networks. In *Advances in Neural Information Processing Systems*, pp 2692–2700.

[2015c] Vinyals O., Kaiser L., Koo T., Petrov S., Sutskever I., Hinton G., Grammar as a foreign language. Google.

[1967] Viterbi A.J., Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In *IEEE Transactions on Information Theory*.

[2010] Vivekanandan P., Nedunchezhian R., A fast genetic algorithm for mining classification rules in large datasets. *International Journal on Soft Computing*, **1**, (1), pp 10–20.

[1944] Von Neumann J., Morgenstern O., Theory of games and economic behavior. Princeton: Princeton University Press, second edition: Princeton UP, 1947.

[1987] Waibel A., Phoneme recognition using time-delay neural networks. Meeting of the Institute of Electrical, Information and Communication Engineers (IEICE). Tokyo, Japan.

[1989] Waibel A., Hanazawa T., Hinton G., Shikano K., Lang K.J., Phoneme recognition using time-delay neural networks. IEEE Transactions on Acoustics, Speech, and Signal Processing, Volume 37, (3), pp 328–339, March 1989.

[1996] Walsh P., Ryan C., Paragen: A novel technique for the autoparallelisation of sequential programs using genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp 406–409, Stanford University, CA, USA. MIT Press.

[2008] Wang W., Nie S., The performance of several combining forecasts for stock index. in *2008 International Seminar on Future Information Technology and Management Engineering*, pp 450–455.

[2009] Wang Y., Liu L., Gu R., Analysis of efficiency for Shenzhen stock market based on multifractal detrended fluctuation analysis. *International Review of Financial Analysis*, **18**, pp 271–276.

[2011] Wang Y., Wei Y., Wu C., Analysis of the efficiency and multifractality of gold markets based on multifractal detrended fluctuation analysis. *Physica A: Statistical Mechanics and its Applications*, **390**, pp 817–827.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2011b] Wang Y., Wu C., Pan Z., Multifractal detrending moving average analysis on the US Dollar exchange rates. *Physica A*, **390**, pp 3512–3523.

[2016] Wang Y., Zhang L., van de Weijer J., Ensembles of generative adversarial networks. Working Paper, arXiv:1612.00991.

[2017] Wang Y., Applications of recurrent neural network on financial time series. Quant Finance Ltd, MSc Thesis, Department of Mathematics, Imperial College London.

[2017] Wang Z., Hamza W., Florian R., Bilateral multi-perspective matching for natural language sentences. arXiv preprint arXiv:1702.03814.

[2018] Wang C., Xu C., Yao X., Tao D., Evolutionary generative adversarial networks. Working Paper, arXiv:1803.00657.

[2018b] Wang A., Singh A., Michael J., Hill F., Levy O., Bowman S.R., Glue: A multi-task benchmark and analysis platform for natural language understanding. Working Paper, arXiv:1804.07461.

[2019] Wang S., Zeng Y., Liu X., Zhu E., Yin J., Xu C., Kloft M., Effective end-to-end unsupervised outlier detection via inlier priority of discriminative network. In *NeurIPS*, pp 5960–5973.

[2019b] Wang A., Hula J., Xia P., Pappagari R., McCoy R.T., Patel R., Kim N., Tenney I., Huang Y., etc. Can you tell me how to get past sesame street? Sentence-level pretraining beyond language modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp 4465–4476.

[2019c] Wang L., Zhao W., Jia R., Li S., Liu J., Denoising based sequence-to-sequence pre-training for text generation. Working Paper, arXiv:1908.08206v1.

[2020] Wang B., Jay Kuo C-C., SBERT-WK: A sentence embedding method by dissecting BERT-based word models. arXiv:2002.06652v2.

[1989] Watkins C.J.C.H., Learning from delayed rewards. Ph.D. Thesis, Kings College, Cambridge, UK.

[1992] Watkins C.J.C.H., Dayan P., Technical note: Q-Learning. *Machine Learning*, **8**, (3-4), pp 279–292.

[1998] Watts D.H., Strogatz S.H., Collective dynamics of small-world networks. *Nature*, **393**, (4), pp 440–442.

[2003] Wegner P., Reilly E.D., Encyclopedia of computer science. Chichester, UK, John Wiley and Sons.

[2018] Wehmeyer C., Noe F., Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics. *J. Chem. Phys.*, 148(24):241703.

[2008] Weis M., Naumann F., Space and time scalability of duplicate detection in graph data. Technical Report 25, Hasso-Plattner-Institut.

[2008] Wendt H., Contributions of wavelet leaders and bootstrap to multifractal analysis: Images, estimation performance, dependence structure and vanishing moments. Confidence intervals and hypothesis tests. Docteur de l'Universite de Lyon, Ecole Normale Superieure de Lyon, Traitement du Signal - Physique.

[1974] Werbos P., Beyond regression: New tools for prediction and analysis in the behavioral sciences. PhD thesis, Harvard University.

[1990] Werbos P., Backpropagation through time: What it does and how to do it. In *Proceedings of the IEEE*, **78**, (10), pp 1550–1560.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1996] Whigham P.A., Search bias, language bias, and genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, Genetic Programming 1996: *Proceedings of the First Annual Conference*, pp 230–237, Stanford University, CA. MIT Press.

[1990] White H., Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings. *Neural Networks*, **3**, pp 535–550.

[2000] White H., A reality check for data snooping. *Econometrica*, **68**, pp 1097–1127.

[1990] Whitley D., Starkweather T., Bogart C., Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, **14**, (3), pp 347–361.

[1999] Whitley D., Rana S., Heckendorn R.B., The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, **7**, pp 33–48.

[1960] Widrow B., Hoff M.E., Adaptive switching circuits. In *WESCON Convention Record*, Part IV, pp 96–104.

[2005] Wiestra D., Gomez F., Schmidhuber J., Modeling systems with internal state Evolino. In GECCO, pp 1795–1802.

[1958] Wigner E.P., Characteristics vectors of bordered matrices with infinite dimensions II. *Annals of Mathematics*, **65**, (2), pp 203–207.

[1978] Wilder W., New concepts in technical trading systems. Trend Research, Greensboro, NC.

[1989] Williams R.J., Zipser D., A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, **1**, (2), pp 270–280.

[1990] Williams R.J., Peng J., An efficient gradient-based algorithm for online training of recurrent network trajectories. *Neural Computation*, **2**, (4), pp 490–501.

[1992] Williams R.J., Zipser D., Gradient-based learning algorithms for recurrent networks and their computational complexity. in Back-propagation: Theory, architectures and applications. NJ:Erlbaum, edt. Y.Chauvin and D.E.Rumelhart, chapter 13, pp 433–486.

[1992] Williams R.J., Simple statistical gradient following algorithms for connectionnist reinforcement learning. *Machine Learning*, **8**, Issue 3-4, pp 229–256.

[2018] Williams A., Nangia N., Bowman S., A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Volume 1 (Long Papers), pp 1112–1122. Association for Computational Linguistics.

[2012] Wilson C., Concave functions of a single variable. Mathematics for Economics, New York University.

[1991] Winkler W.E., Thibaudeau Y., An application of the Fellegi-Sunter model of record linkage to the 1990 US decennial Census. Technical Report Statistical Research Report Series RR91/09, US Bureau of the Census, Washington, D.C.,.

[1993] Winkler W.E., Improved decision rules in the Felligi-Sunter model of record linkage. Technical Report Statistical Research Report Series RR93/12, US Bureau of the Census, Washington, D.C.,.

[2002] Winkler W.E., Methods for record linkage and Bayesian networks. Technical Report Statistical Research Report Series RRS2002/05, US Bureau of the Census, Washington, D.C.,.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2006] Winkler W.E., Overview of record linkage and current research directions. Technical Report Statistical Research Report Series RRS2006/02, US Bureau of the Census, Washington, D.C.,.

[2011] Witkowski B., Building a solver for optimisation problems. BSc Thesis, University of Science and Technology in krakow.

[1977] Witten I.H., An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, **34**, pp 286–295.

[2005] Witten J.H., Frank E., Data mining: Practical machine learning tools and techniques. Morgan Kaufmann, 2 edition.

[1992] Wolpert D.H., Original contribution: Stacked generalization. *Neural Networks*, **5**, (2), pp 241–259.

[2005] Wong M.L., Evolving recursive programs by using adaptive grammar based genetic programming. *Genetic Programming and Evolvable Machines*, **6**, (4), pp 421–455.

[2006] Wong Y.W., Mooney R.J., Learning for semantic parsing with statistical machine translation. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, Association for Computational Linguistics.

[2009] Wong W., Liu W., Bennamoun M., Acquiring semantic relations using the Web for constructing lightweight ontologies. In *Proc. 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*.

[1991] Wright A.H., Genetic algorithms for real parameter optimization. in Foundation of Genetic Algorithms, ed. G. Rawlins, *First Workshop on the Foundation of Gen. Alg. and Classified Systems*, Los Altos, CA, pp 205-218.

[2016] Wu Y., Schuster M., Chen Z., Le Q.V., Norouzi M., Macherey W., etc. Google's neural machine translation system: Bridging the gap between human and machine translation. Working Paper, arXiv:1609.08144v2.

[2018] Wu H., Mardt A., Pasquali L., Noe F., Deep generative Markov state models. In *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, Montreal, Canada.

[2008] Wyffels F., Schrauwen B., Verstraeten D., Stroobandt D., Band-pass reservoir computing. In 2008 *IEEE International Joint Conference on Neural Networks*, pp 3204–3209.

[2009] Wyffels F., Schrauwen B., Design of a central pattern generator using reservoir computing for learning human motion. Electronics and Information Systems Department, Ghent University, Belgium.

[1931] Wynn-Williams C.E., The use of thyratrons for high speed automatic counting of physical phenomena. *Proceedings of the Royal Society A*, **132**, (819), pp 295–310.

[1995] Xu L., Jordan M.I., Hinton G.E., An alternative model for mixtures of experts. In G. Tesauro, D.S. Touretzky, and T.K. Leen, editors, *Advances in Neural Information Processing Systems 7*, MIT Press, pp 633–640. Cambridge, MA, 1995.

[1996] Xu L., Jordan M.I., On convergence properties of the EM algorithm for Gaussian mixtures. *Neural Computation*, **8**, (1), pp 129–151.

[2007] Xu Y., Yang L., Haykin S., Decoupled echo state networks with lateral inhibition. *Neural Networks*, **20**, (3), pp 365–376.

[2015] Yang Z., Yang D., Dyer C., He X., Smola A., Hovy E., Hierarchical attention networks for document classification. Working Paper.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2019] Yang Z., Dai Y., Yang Y., Carbonell J., Salakhutdinov R., Le Q.V., XLNet: Generalized autoregressive pretraining for language understanding. In *Proceedings of Advances in Neural Information Processing Systems*.

[1996] Yao J.T., Poh H.L., Equity forecasting: A case study on the KLSE index. *Neural Networks in Financial Engineering, Proceedings of 3rd International Conference on Neural Networks in the Capital Markets*, Oct 1995, London, A-P.N. Refenes, Y. Abu-Mostafa, J. Moody and A. Weigend, (eds.), World Scientific, pp 341–353.

[1997] Yao X., Liu Y., A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, **8**, (3), pp 694–713.

[1998] Yao J.T., Tan C.L., A study on training criteria for financial time series forecasting. Working Paper.

[1999] Yao X., Evolving artificial neural networks. In *Proceedings of the IEEE*, **87**, (9), pp 1423–1447.

[2000] Yao J.T., Tan C.L., Time dependent directional profit model for financial time series forecasting. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, July 2000, Como, Italy, **5**, pp 291–296.

[2000b] Yao J.T., Tan C.L., Option price forecasting using neural networks. *Omega*, **28**, pp 455–466.

[2006] Yasuda T., Nakamura K., Kawahara A., Tanaka K., Neural network with variable type connection weights for autonomous obstacle avoidance on a prototype of six-wheel type intelligent wheelchair. *International Journal of Innovative Computing, Information and Control*, **2**, (5), pp 1165–1177.

[2010] Yazdi H.S., Pourreza R., Unsupervised adaptive neural-fuzzy inference system for solving differential equations. *Applied Soft Computing*, **10**, (1), pp 267-275.

[2008] Yin X., Han J., Yu P.S., Truth discovery with multiple conflicting information providers on the web. *IEEE Transactions on Knowledge and Data Engineering*, **20**, (6), pp 796–808.

[2019] Yogatama D., de Masson d'Autume C., Connor J., Kocisky T., Chrzanowski M., Kong L., Lazaridou A., Ling W., Yu L., Dyer C., Blunsom P., Learning and evaluating general linguistic intelligence. Working Paper, arXiv:1901.11373v1.

[2015] Yosinski J., Clune J., Nguyen A., Fuchs T., Lipson H., Understanding neural networks through deep visualization. arXiv preprint arXiv:1506.06579.

[2004] Yu T., Chen S-H., Using genetic programming with lambda abstraction to find technical trading rules. In *Computing in Economics and Finance*, University of Amsterdam.

[2013] Yu D., Seltzer M.L., Li J., Huang J-T., Seide F., Feature learning in deep neural networks-studies on speech recognition tasks. arXiv preprint arXiv:1301.3605.

[2009] Yuan Y., Zhuang X-T., Jin X., Measuring multifractality of stock price fluctuation using multifractal detrended fluctuation analysis. *Physica A*, **388**, (11), pp 2189–2197.

[1900] Yule G., On the association of attributes in statistics. *Philosophical Transactions of the Royal Society of London*, **194**, pp 257–319.

[1965] Zadeh L.A., Fuzzy sets. *Information and Control*, **8**, (3), pp 338–353.

[1973] Zadeh L.A., Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Trans. Systems*, Man and Cybernetics, 3, pp 28–44.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1994] Zadeh L.A., Fuzzy logic, neural networks, and soft computing. *Communications of the ACM*, **37**, (3), pp 77–84.

[2002] Zaharie D., Critical values for the control parameters of differential evolution algorithms. in R. Matousek, P. Osmera, eds., *Proceedings of MENDEL 2002, 8th International Conference on Soft Computing*, Brno University of Technology, Faculty of Mechanical Engineering, pp 62–67, Institute of Automation and Computer Science.

[2012] Zeiler M.D., Adadelta: An adaptive learning rate method. arXiv preprint arXiv:1212.5701.

[2014] Zen H., Senior A., Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

[2018a] Zenati H., Foo C.S., Lecouat B., Manek G., Chandrasekhar V.R., Efficient gan-based anomaly detection. arXiv:1802.06222.

[2018b] Zenati H., Romain M., Foo C.S., Lecouat B., Chandrasekhar V.R., Adversarially learned anomaly detection. In *ICDM*, IEEE, pp 727–736.

[2006] Zeng F., Zhang Y., Stock index prediction based on the analytical center of version space. *Advances in Neural Networks*, **3973**, pp 458–463.

[1988] Zhang W., Tanida J., Kazuyoshi I., Ichioka Y., Shift-invariant pattern recognition neural network and its optical architecture. In *Proceedings of Annual Conference of the Japan Society of Applied Physics*.

[1990] Zhang W., Kazuyoshi I., Tanida J., Ichioka Y., Parallel distributed processing model with local space-invariant interconnections and its optical architecture. *Applied Optics*, **29**, (32), pp 4790–4797.

[2007] Zhang X., Chen Y., Yang J.Y., Stock index forecasting using pso based selective neural network ensemble. in *International Conference on Artificial Intelligence*, pp 260–264.

[2012] Zhang H., Rangaiah G.P., An efficient constraint handling method with integrated differential evolution for numerical and engineering optimization. *Computers and Chemical Engineering*, **37**, pp 74–88.

[2018] Zhang Z., Lui S., Li M., Zhou M., Chen E., Bidirectional generative adversarial networks for neural machine translation. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pp 190–199.

[2020] Zhang Z., Wu Y., Zhao H., Li Z., Zhang S., Zhou X., Zhou X., Semantics-aware BERT for language understanding. Working Paper, arXiv:1909.02209v3.

[2015] Zhao L., Chen J., Chen F., Wang W., Lu C-T., Ramakrishnan N., Simnest: Social media nested epidemic simulation via online semisupervised deep learning. In *Data Mining (ICDM)*, 2015 IEEE International Conference on. IEEE, pp 639–648.

[2014] Zhaoping L., Understanding vision: Theory, models, and data. United Kingdom, Oxford University Press.

[2019] Zheng P., Yuan S., Wu X., Li J., Lu A., One-class adversarial nets for fraud detection. In *AAAI*, pp 1286–1293.

[2012] Zhou Z-H., Ensemble methods: Foundations and algorithms. Chapman & Hall, Machine Learning & Pattern Recognition Series.

[2008] Zhu X., Wang H., Xu L., Li H., Predicting stock index increments by neural networks: The role of trading volume under different horizons. *Expert Syst. Appl.*, **34**, (4), pp 3043–3054.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2017] Zhu J-Y., Park T., Isola P., Efros A.A., Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, pp 2223–2232.

[2005] Zielinski K., Peters D., Laur R., Stopping criteria for single-objective optimization. In Proceedings of the Third International Conference on Computational Intelligence, Robotics and Autonomous Systems, Singapore.

[2006] Zielinski K., Laur R., Constrained single-objective optimization using differential evolution. in *2006 IEEE Congress on Evolutionary Computation*, Vancouver, Canada, pp 223–230.

[2007] Zielinski K., Laur R., Stopping criteria for a constrained single-objective particle swarm optimization algorithm. *Informatica*, **31**, pp 51–59.

[2008] Zielinski K., Laur R., Stopping criteria for differential evolution in constrained single-objective optimization. in Chakrabotry, Ed., Advances in Differential Evolution, **143**, Springer-Verlag, Berlin, pp 111–138.

[2006] Zimmermann H.G., Grothmann R., Schaefer A.M., Tietz C., Identification and forecasting of large dynamical systems by dynamical consistent neural networks. in *New Directions in Statistical Signal Processing: From systems to brain*. MIT Press, edt S.Haykin, J.Principe, T.Sejnowski, J.McWhirter, pp 203–242.

[2003] Zinkevich M., Online convex programming and generalized infinitesimal gradient ascent. in *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, pp 928–936.

[2009] Zinkevich M., Smola A., Langford J., Slow learners are fast. In Y. Bengio, D. Schuurmans, J. Lafferty, C.K.I. Williams, and A. Culotta, editors, Advances in **Neural Information Processing Systems 22**, pp 2331–2339.

[2016] Zoph B., Le Q.V., Neural architecture search with reinforcement learning. arXiv:1611.01578, 2016.

[2018] Zou H-L., Yu Z-G., Anh V., Ma Y-L., From standard alpha-stable Levy motions to horizontal visibility networks: Dependence of multifractal and Laplacian spectrum. *Journal of Statistical Mechanics: Theory and Experiment*, **2018**.

[2007] Zunino L., Tabak B.M., Perez D.G., Garavaglia M., Rosso O.A., Inefficiency in Latin-American market indices. *The European Physical Journal B*, **60**, (1), pp 111–121.

[2008] Zunino L., Tabak B.M., Figlioa A., Perez D.G., Garavaglia M., Rosso O.A., A multifractal approach for stock market inefficiency. *Physica A*, **387**, pp 6558–6566.

[1993] Zweig M.H., Campbell G., Receiver-operating characteristic (ROC) plots: A fundamental evaluation tool in clinical medicine. *Clinical Chemistry*, **39**, (8), pp 561–577.

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Electronic copy available at: https://ssrn.com/abstract=3307566