

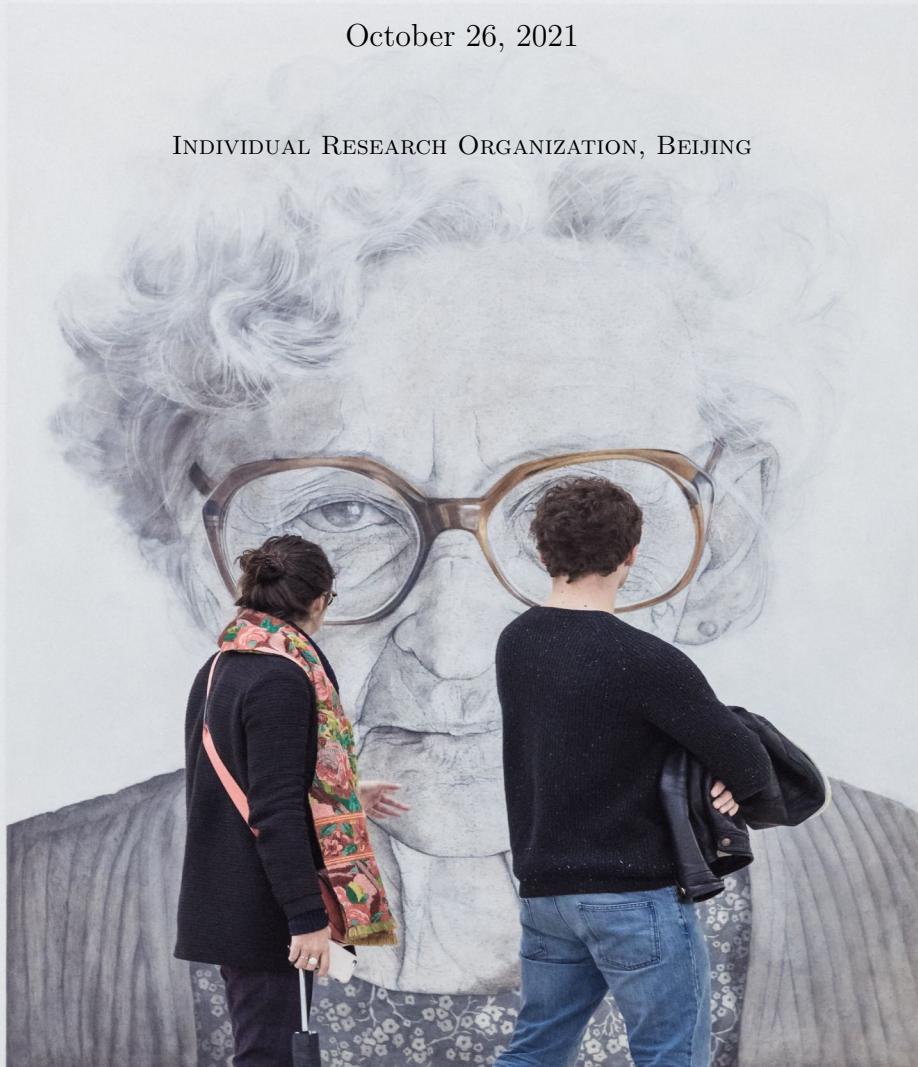
Notes on Machine Learning

Tong Jia

cecilio.jia@gmail.com

October 26, 2021

INDIVIDUAL RESEARCH ORGANIZATION, BEIJING



"This is a clever quote or perhaps a dedication."

PERSONAL RESEARCH INTERNSHIP, BEIJING

GITHUB.COM/CECILIO-JIA

Not for commercial usage, October 26, 2021

Content

I Machine Learning Basic	5
1 Introduction	6
1.1 Machine Learning System	6
1.1.1 Architecture of End-to-End Machine learning System	6
1.1.2 Ecosystem of Machine Learning Production	7
1.2 Framework of Machine Learning	8
1.3 Framework of Model in Practice	9
1.4 Large-Scale Machine Learning Training System	10
1.4.1 Reference	10
2 Dataset	11
2.1 Dataset Representation	11
2.2 Independent Variable	12
2.2.1 Category features	12
2.2.2 Numerical features	13
2.3 Dependent Variable	14
2.3.1 Imbalanced dataset in classification task	14
2.4 Dataset Preprocessing	17
2.4.1 Image preprocessing	17
2.4.2 Text preprocessing	18
2.5 Analysis of Dataset (未完成)	19
2.5.1 Significance Analysis	19
2.5.2 Correlation Analysis	19
2.5.3 Principal Component Analysis (PCA)	20
2.5.4 Linear Discriminant Analysis (LDA)	21
2.6 Summary of Dataset	22
3 Objective Function	23
3.1 Expected risk & Empirical risk	23
3.1.1 Expected risk minimization	23
3.1.2 Empirical risk minimization	23
3.2 Loss Function for Regression	24
3.2.1 Mean squared error	24
3.2.2 Mean absolute error	24
3.2.3 Huber loss	24
3.3 Loss Function for Classification	26
3.3.1 Hinge loss	26
3.3.2 Log-likelihood	27
3.3.3 Kullback–Leibler divergence	27

3.3.4	Cross entropy	29
3.3.5	Focal loss	30
3.3.6	Gradient Harmonized Mechanism (GHM)	34
3.3.7	Softmax + Cross entropy loss	37
3.3.8	Large-Margin Softmax + Cross entropy loss (L-Softmax)	41
3.3.9	Additive Margin Softmax + Cross entropy loss (AM-Softmax)	47
3.3.10	Angular-Softmax + Cross entropy loss (A-Softmax)	48
3.3.11	ArcFace + Cross entropy loss	51
3.3.12	Triplet loss	52
3.3.13	COCO loss	56
3.3.14	Center loss	58
3.3.15	Center Invariant loss	59
3.3.16	Wasserstein distance and the Kantorovich-Rubinstein duality	60
3.4	Regularization	63
3.4.1	L1 regularization (Lasso regularization)	63
3.4.2	L2 regularization (Ridge regularization)	64
3.4.3	Mini-batch aware regularization	65
3.4.4	Summary of regularization	66
4	Optimization Algorithms	67
4.1	Overview	67
4.2	Gradient Descent Variants	68
4.3	Gradient Descent Optimization Algorithms	70
4.3.1	SGD	70
4.3.2	Momentum	71
4.3.3	Nesterov momentum	72
4.3.4	AdaGrad	74
4.3.5	AdaDelta	76
4.3.6	RMSProp	77
4.3.7	Adam	78
4.3.8	AdaBound	85
4.3.9	RAdam (未完成)	87
4.3.10	SWATS: Improving Generalization Performance by Switching from Adam to SGD	88
4.4	Online Learning Optimization Algorithms	90
4.4.1	FTRL (未完成)	90
4.5	Constrained Optimization Algorithms (未完成)	91
4.5.1	Proxy-Lagrangian Optimization	91
4.6	Summary of Gradient Based Optimizers	92
4.7	References	93

5 Evaluation Metrics	94
5.1 Regression Task	94
5.1.1 Mean Absolute Error (MAE)	94
5.1.2 Mean Squared Error (MSE)	94
5.1.3 Root Mean Squared Error (RMSE)	95
5.1.4 Mean Absolute Percentage Error (MAPE)	95
5.1.5 Coefficient of determination (R^2)	95
5.2 Classification Task	96
5.2.1 Log Loss	96
5.2.2 Confusion Matrix	96
5.2.3 Accuracy	96
5.2.4 Precision	96
5.2.5 Recall (Sensitivity)	96
5.2.6 Specificity	97
5.2.7 F1-score	97
5.2.8 AUC & ROC	97
5.3 Generative Adversarial Task (未完成)	99
5.3.1 Inception Score (IS) (未完成)	99
5.3.2 Fréchet Inception Distance (FID)	99
5.3.3 Mode Score (MS)	99
5.3.4 Kernel Maximum Mean Discrepancy (Kernel MMD)	99
5.3.5 Wasserstein Distance (WD)	99
6 Model, Feature & Hyper-parameters Selections	100
6.1 Model Selection	100
6.1.1 Overfitting & Underfitting	100
6.1.2 Bias-Variance Decomposition	101
6.2 Feature Selection	104
6.2.1 Filter methods	104
6.2.2 Wrapper methods	104
6.2.3 Embedded methods	104
6.3 Hyper-parameters Selection	105
6.3.1 Model validation strategies	105
II Machine Learning Model and Theory	106
7 Logistic Regression	107
7.1 Logistic Regression	107
7.1.1 Model prediction	107
7.1.2 Model training	107
7.1.3 Model implement	110
7.2 Softmax Regression	116

7.2.1	Model prediction	116
7.2.2	Model training	117
7.2.3	Summary of softmax regression	119
7.2.4	Reference	119
8	Naive Bayes	120
8.1	Naive Bayes Classification	120
8.1.1	Model prediction	120
8.1.2	Model training	120
8.1.3	Naive bayes note	121
8.2	Logistic Regression VS Naive Bayes	122
9	Graphical Models	123
9.1	Hidden Markov Models (HMM)	123
9.1.1	Introduction (未完成)	123
9.1.2	Model formulation	123
9.1.3	Reference	124
9.2	Conditional Random Fields (CRF)	125
9.2.1	Model formulation	125
9.2.2	Reference	126
10	Time Series Models	127
10.1	ARIMA (未完成)	127
10.1.1	Autoregressive models (AR)	127
10.1.2	Moving average models (MA)	127
10.1.3	Autoregressive moving average models (ARMA)	127
10.2	Online ARIMA Algorithms for Time Series Prediction	128
10.2.1	Introduction	128
10.3	Prophet (未完成)	129
10.4	Modeling Long and Short-Term Temporal Patterns with Deep Neural Networks (未完成)	130
11	Support Vector Machine	131
11.1	Hard-margin SVM Classification	131
11.1.1	Model prediction	131
11.1.2	Model training	131
11.1.3	Summary of hard-margin svm	135
11.2	Soft-margin SVM Classification	136
11.2.1	Model prediction	136
11.2.2	Model training of dual formulation	136
11.2.3	Model training of hinge loss	138
11.3	Kernel Trick for Nonlinear Classification	140

12 Decision Tree	142
12.1 Classification Tree	142
12.1.1 Model prediction	142
12.1.2 Model training	143
12.1.3 Model pruning	147
12.2 Regression Tree	148
12.2.1 Model training	148
13 Ensemble Learning	149
13.1 Random Forest	149
13.1.1 Model formulation	149
13.1.2 Random forest classification	149
13.1.3 Random forest regression	149
13.2 Adaboost	150
13.2.1 Adaboost for binary classification	150
13.2.2 Adaboost for multi classification	153
13.2.3 Adaboost for regression	154
13.3 Gradient Boosting Machine	155
13.3.1 Model formulation	155
13.3.2 GBM regression	155
13.3.3 GBM classification	156
13.4 XGBoost	157
13.4.1 Model prediction	157
13.4.2 Model training	157
13.4.3 Model pruning	162
13.5 LightGBM	163
13.5.1 Introduction	163
13.5.2 Model training	164
13.5.3 Reference	165
13.6 CatBoost (未完成)	166
13.6.1 Introduction	166
13.6.2 Model formulation	166
13.6.3 Reference	167
13.7 NGBoost (未完成)	168
13.7.1 Introduction	168
14 Dimensionality Reduction	169
14.1 t-Distributed Stochastic Neighbor Embedding (t-SNE)	169
14.1.1 Introduction	169
14.1.2 Model formulation	170
14.1.3 Reference	172

III Deep Learning Model and Theory	173
15 Dense Neural Networks (DNNs)	174
15.1 Fully Connection (FC)	174
15.1.1 Overview	174
15.1.2 Derivation of forward-propagation and back-propagation in fully-connected layer (未完成)	174
16 Convolutional Neural Networks (CNNs)	177
16.1 Convolution (未完成)	177
16.1.1 Convolution as matrix multiplication	177
16.1.2 Derivation of forward-propagation and back-propagation in CNN (未完成, 重要) . .	178
16.1.3 Reference	179
16.1.4 Summary	180
16.2 Transposed Convolution (未完成)	181
16.2.1 Derivation of forward-propagation and back-propagation in Transposed Convolution (未完成, 重要)	181
16.2.2 Reference	182
16.3 LeNet5	183
16.3.1 Model formulation	183
16.3.2 Model implement	183
16.4 AlexNet	191
16.4.1 Model formulation	191
16.4.2 Model implement	191
16.4.3 Reference	196
16.5 VGG	197
16.5.1 Model formulation	197
16.6 GoogLeNet	198
16.6.1 Model formulation	198
16.7 ResNet	199
16.7.1 Model formulation	199
16.8 DenseNet	201
16.8.1 Model formulation	201
17 Recurrent Neural Networks (RNNs)	202
17.1 Introduction	202
17.2 Vanilla Recurrent Neural Network (RNN)	203
17.2.1 Derivation of forward-propagation in Vanilla RNN	203
17.2.2 Derivation of back-propagation through time (BPTT) in Vanilla RNN	203
17.2.3 Vanishing gradient problem in RNN	205
17.2.4 References	207
17.3 Long Short-Term Memory Neural Network (LSTM)	208
17.3.1 Derivation of forward-propagation in LSTM	208

17.3.2 Derivation of back-propagation through time (BPTT) in LSTM	211
17.3.3 Model implement	213
17.4 Gated Recurrent Unit (GRU)	216
17.4.1 Derivation of forward-propagation in GRU	216
17.5 Bidirectional LSTM (Bi-LSTM)	217
17.5.1 Derivation of forward-propagation in Bi-LSTM	217
17.5.2 Model implement	217
17.6 Multi Layers Recurrent Neural Networks	221
17.6.1 Derivation of forward-propagation in Multi-layer RNNs	221
17.6.2 Model implement	221
17.7 Bi-LSTM with Multi Layers LSTMs	225
17.7.1 Forward propagation	225
18 Sequence to Sequence Learning (Seq2Seq)	226
18.1 Transformer (未完成)	226
18.1.1 Introduction	226
18.1.2 Model formulation	227
18.1.3 References	237
18.2 Stabilizing Transformers for Reinforcement Learning (未完成)	238
19 Generative Adversarial Networks (GANs)	239
19.1 Vanilla GAN	239
19.1.1 Introduction	239
19.1.2 Model formulation	240
19.2 DCGAN (未完成)	243
19.2.1 Introduction	243
19.2.2 Model formulation	243
19.3 Mode Seeking GANs (MSGANs)	244
19.3.1 Introduction	244
19.3.2 Model formulation	246
19.4 Wasserstein GAN (WGAN)	249
19.4.1 Introduction (未完成)	249
19.4.2 Wasserstein distance and the Kantorovich-Rubinstein duality (未完成)	254
19.4.3 Model formulation	255
19.4.4 Reference	257
19.5 Wasserstein GAN with Gradient Penalty (WGAN-GP)	258
19.5.1 Introduction	258
19.5.2 Model formulation	258
19.6 Wasserstein GAN with Lipschitz Penalty (WGAN-LP)	259
19.6.1 Introduction	259
19.6.2 Model formulation	259
19.7 Wasserstein GAN with Wasserstein Gradient Regularization (WWGAN)	260

19.7.1 Introduction	260
19.8 Wasserstein GAN with Consistency Term (CT-GAN) (未完成)	261
19.8.1 Introduction	261
19.9 Virtual Adversarial Lipschitz Regularization (未完成)	262
19.9.1 Introduction	262
19.9.2 model formulation	262
19.10 Survey and Summary (未完成)	263
19.10.1 Introduction	263
19.10.2 Architecture-variant GANs	264
19.10.3 Loss-variant GANs	265
19.10.4 Applications of GANs	266
19.10.5 Personal ideas of GANs	267
20 Variational Autoencoders (VAEs)	268
20.1 An Introduction to Variational Autoencoders (未完成)	268
20.1.1 Introduction	268
21 Graph Neural Networks (GNNs)	269
21.1 Overview of GCNs	269
21.1.1 Spectral-based GCNs	277
21.1.2 Spatial-based GCNs	279
21.2 DeepWalk: Online Learning of Social Representations (未完成)	282
21.2.1 Introduction	282
21.2.2 Model formulation	283
21.3 LINE: Large-scale Information Network Embedding (未完成)	284
21.3.1 Introduction	284
21.4 Node2vec: Scalable Feature Learning for Networks (未完成)	285
21.4.1 Introduction	285
21.5 SDNE: Structural Deep Network Embedding (未完成)	286
21.5.1 Introduction	286
21.6 Struc2vec: Learning Node Representations from Structural Identity (未完成)	287
21.6.1 Introduction	287
21.7 Survey and Summary (未完成)	288
21.7.1 Introduction	288
22 Attention Mechanisms	289
22.1 Self-Attention	289
22.1.1 Self-attention mechanisms in Computer Vision	289
22.1.2 Self-attention mechanisms in Natural Language Processing	289
22.1.3 Reference	289

23 Network Optimization and Regularization	290
23.1 Exponential Moving Average (EMA)	290
23.2 Activation Function	291
23.2.1 Logistic function (Sigmoid function)	291
23.2.2 Tanh function	292
23.2.3 ReLU function	294
23.2.4 Leaky-ReLU function	295
23.2.5 Parametric-ReLU function	295
23.3 Network Initialization	296
23.3.1 Xavier initialization	296
23.3.2 He initialization	301
23.4 Network Normalization	302
23.4.1 Batch Normalization (BN)	302
23.4.2 Layer Normalization (LN)	306
23.4.3 Instance Normalization (IN)	307
23.4.4 Group Normalization (GN)	311
23.4.5 Batch-Instance Normalization (BIN)	312
23.4.6 Local Response Normalization (LRN)	315
23.4.7 Gradient Normalization (Gradient Clipping)	316
23.5 Network Regularization	317
23.5.1 Label Smoothing Regularization (LSR)	317
23.6 Dropout	318
23.6.1 Dropout of DNN	318
23.6.2 Dropout of CNN	320
23.6.3 Dropout of RNN	321
23.7 Learning Rate Schedules	322
23.7.1 Learning Rate Decay	322
23.7.2 Learning Rate Warmup	323
23.7.3 Periodic Learning Rate Adjustment	324
23.7.4 Linear Scaling Learning Rate	325
23.8 Summary Tricks of Training Neural Networks	326
23.8.1 Tricks for DNN	326
23.8.2 Tricks for CNN	327
23.8.3 Tricks for RNN	328
IV Reinforcement Learning Model and Theory	329
24 Reinforcement Learning Basic	330
24.1 Markov Decision Processes (MDPs)	330
24.1.1 Infinite-horizon discounted MDPs	330
24.1.2 Interaction protocol	331

24.1.3 Policy and value	332
24.1.4 Bellman equations	333
24.1.5 Bellman optimality equations	336
24.1.6 Summary	337
24.1.7 Reference	338
24.2 Semi-Markov Decision Processes (Semi-MDPs) (未完成)	339
24.3 Partially Observable Markov Decision Processes (POMDPs)	340
24.3.1 Definition	340
24.3.2 Reference	341
24.4 Dynamic Programming (DP) in MDPs	342
24.4.1 Value iteration	342
24.4.2 Policy iteration	343
24.4.3 Reference	344
24.5 Applications of Reinforcement Learning Algorithms	345
24.5.1 Survey	345
24.5.2 Finance	345
24.5.3 Healthcare	345
24.5.4 Robotics	345
24.5.5 Transportation	345
24.5.6 Recommender Systems	345
24.6 Frameworks of Reinforcement Learning Algorithms	346
24.6.1 Commerical	346
24.6.2 Educational	346
24.7 Summary of Reinforcement Learning Algorithms	347
24.7.1 Model-free RL	347
25 Policy-Based Deep Reinforcement Learning Algorithms	348
25.1 Vanilla Policy Gradient (VPG)	348
25.1.1 Introduction	348
25.1.2 Model formulation	349
25.1.3 Forms of the policy gradient summary	353
25.1.4 Reference	357
25.2 Optimality and Approximation with Policy Gradient Methods in Markov Decision Processes	358
25.2.1 Introduction	358
25.2.2 Reference	359
25.3 Trust Region Policy Optimization (TRPO)	360
25.3.1 Introduction	360
25.3.2 Model formulation	361
25.3.3 Reference	367
25.4 Proximal Policy Optimization (PPO)	368
25.4.1 Introduction	368
25.4.2 Model formulation	371

25.4.3 Reference	376
25.5 Asynchronous Methods for Deep Reinforcement Learning (A3C)	377
25.5.1 Introduction	377
25.6 Sample Efficient Actor-Critic with Experience Replay (ACER)	378
25.6.1 Introduction	378
25.7 Deterministic Policy Gradient Algorithms (DPG)	379
25.7.1 Introduction	379
25.8 Continuous Control with Deep Reinforcement Learning (DDPG)	380
25.8.1 Introduction	380
25.8.2 Model formulation	381
25.8.3 Reference	384
25.9 Distributed Distributional Deterministic Policy Gradients (D4PG)	385
25.9.1 Introduction	385
25.10 Addressing Function Approximation Error in Actor-Critic Methods (TD3)	386
25.10.1 Introduction	386
25.10.2 Model formulation	387
25.10.3 Reference	388
25.11 Soft Actor-Critic Algorithms and Applications (SAC) (未完成)	389
25.11.1 Introduction	389
25.11.2 Model formulation	391
25.11.3 Understanding the Impact of Entropy on Policy Optimization	395
25.11.4 Reference	396
25.12 Boosting Soft Actor-Critic: Emphasizing Recent Experience without Forgetting the Past	397
25.12.1 Introduction	397
25.12.2 Model formulation (未完成)	399
25.13 Action Robust Reinforcement Learning and Applications in Continuous Control	400
25.13.1 Introduction	400
25.14 Remember and Forget for Experience Replay (ReF-ER)	401
25.14.1 Introduction	401
25.14.2 Model formulation	402
25.15 Dimension-Wise Importance Sampling Weight Clipping for Sample-Efficient Reinforcement Learning	403
25.15.1 Introduction	403
26 Value-Based Deep Reinforcement Learning Algorithms	404
26.1 Deep Q-Network (DQN)	404
26.1.1 Introduction	404
26.1.2 Model formulation	406
26.1.3 Reference	407
26.2 Deep Reinforcement Learning with Double Q-learning (Double DQN)	408
26.2.1 Introduction	408
26.2.2 Model formulation	409

26.3	Dueling Network Architectures for Deep Reinforcement Learning (Dueling DQN)	410
26.3.1	Introduction	410
26.3.2	Model formulation	410
26.4	A Distributional Perspective on Reinforcement Learning (C51) (未完成)	411
26.4.1	Introduction	411
26.4.2	Reference	412
26.5	Prioritized Experience Replay	413
26.5.1	Introduction	413
26.6	Distributed Prioritized Experience Replay	414
26.6.1	Introduction	414
26.7	Rainbow: Combining Improvements in Deep Reinforcement Learning	415
26.7.1	Introduction	415
27	Model-Based Deep Reinforcement Learning	416
27.1	Benchmarking Model-Based Reinforcement Learning	416
27.1.1	Reference	416
28	Multi-Agent Deep Reinforcement Learning	417
28.1	QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning	417
28.1.1	Introduction	417
29	Imitation Learning	418
29.1	A Divergence Minimization Perspective on Imitation Learning Methods	418
29.1.1	Introduction	418
29.1.2	Model formulation	418
29.1.3	Reference	418
29.2	Imitation Learning from Imperfect Demonstration	419
29.2.1	Introduction	419
30	Inverse Reinforcement Learning	420
30.1	Algorithms for Inverse Reinforcement Learning	420
30.1.1	Introduction	420
30.2	Imitation Learning from Imperfect Demonstration	421
30.2.1	Introduction	421
30.3	Discriminator-Actor-Critic: Addressing Sample Inefficiency and Reward Bias in Adversarial Imitation Learning (未完成)	423
30.3.1	Introduction	423
31	Safe Reinforcement Learning	424
31.1	Constrained Policy Optimization	424
31.1.1	Introduction	424
31.2	Safe Exploration in Continuous Action Spaces	425
31.2.1	Introduction	425

31.3 Constrained Cross-Entropy Method for Safe Reinforcement Learning	426
31.3.1 Introduction	426
32 Meta Reinforcement Learning	427
32.1 Meta Reinforcement Learning Overview	427
32.1.1 Introduction	427
V Automated Machine Learning (AutoML) Approach and Theory	428
33 AutoML for Feature Selection	429
33.1 AutoCross: Automatic Feature Crossing for Tabular Data in Real-World Applications (未完成)	429
33.1.1 Introduction	429
34 AutoML for Model Architecture Selection	430
35 AutoML for Loss Function	431
35.1 AM-LFS: AutoML for Loss Function Search (未完成)	431
35.1.1 Introduction	431
36 AutoML for Optimizer Selection	432
37 AutoML for Hyper-Parameter Selection	433
VI Knowledge Graph Model and Theory	434
38 Introduction of Knowledge Graph (未完成)	435
38.1 title	435
VII Model Applications of Computer Vision	436
39 Semantic Segmentation (Image Segmentation)	437
39.1 Fully Convolutional Networks for Semantic Segmentation	437
39.1.1 Introduction	437
39.1.2 Reference	437
40 Image Synthesis	438
40.1 BigGAN (未完成)	438
40.1.1 Introduction	438
40.2 StyleGAN (未完成)	439
40.2.1 Introduction	439

41 Image-to-Image Translation	440
41.1 CycleGAN	440
41.1.1 Introduction	440
41.1.2 Model formulation	442
41.1.3 Reference	447
41.2 Augmented CycleGAN (未完成)	448
41.2.1 Introduction	448
41.3 DualGAN	449
41.3.1 Introduction	449
41.3.2 Model formulation	449
41.4 DiscoGAN	453
41.4.1 Introduction	453
41.4.2 Model formulation	454
42 Image Inpainting	457
42.1 Semantic Image Inpainting with Deep Generative Models (未完成)	457
42.1.1 Introduction	457
43 Object Detection	458
VIII Model Applications of Natural Language Processing	459
44 Word Representation	460
44.1 A Neural Probabilistic Language Model	460
44.1.1 Introduction	460
44.1.2 Model formulation	460
44.2 Word2vec	461
44.2.1 Continuous Bag-of-Word Model (CBOW)	462
44.2.2 Skip-Gram Model	467
44.2.3 Subsampling frequent words	469
44.2.4 Negative sampling	471
44.3 BERT (未完成)	472
44.3.1 Introduction	472
44.3.2 Reference	473
44.4 XLNet (未完成)	474
44.4.1 Introduction	474
45 Text Classification	475
45.1 Convolutional Neural Networks for Sentence Classification	475
45.1.1 Introduction	475
45.1.2 Model formulation	475
45.2 Deep Pyramid Convolutional Neural Networks for Text Categorization	476
45.2.1 Introduction	476

45.2.2 Model formulation	476
46 Semantic Matching	477
46.1 DSSM	477
46.1.1 Introduction	477
46.1.2 Model formulation	478
47 Machine Reading Comprehension	479
47.1 NumNet: Machine Reading Comprehension with Numerical Reasoning (未完成)	479
47.1.1 Introduction	479
48 Sequence Generation	480
48.1 SeqGAN (未完成)	480
48.1.1 Introduction	480
49 Sequence Tagging	481
49.1 Bidirectional LSTM-CRF Models for Sequence Tagging (Bi-LSTM + CRF) (未完成)	481
49.1.1 Introduction	481
IX Model Applications of Autonomous Driving	482
50 Deep Learning Approaches	483
50.1 A Survey of Deep Learning Techniques for Autonomous Driving	483
50.1.1 Reference	483
X Model Applications of Real-Time Bidding	484
51 Deep Reinforcement Learning Approaches	485
XI Model Applications of Recommender System	486
52 Low-Rank Matrix Factorization Approaches	487
52.1 Collaborative Filtering (CF)	488
52.1.1 Introduction	488
52.1.2 Model formulation	488
52.1.3 Characteristics	489
52.2 Singular Value Decomposition (SVD)	490
52.2.1 Introduction	490
52.2.2 Model formulation	490
52.2.3 Characteristics	490
52.3 Matrix Factorization (MF)	491
52.3.1 Introduction	491
52.3.2 Model formulation	491

52.3.3 Characteristics	491
53 Embedding Learning Approaches	493
53.1 Real-time Personalization using Embeddings for Search Ranking at Airbnb	493
53.1.1 Introduction	493
53.1.2 Model training for listing embeddings (Short-term)	493
53.1.3 Model training for user-type & listing-type embeddings (Long-term)	497
53.2 Learning and Transferring IDs Representation in E-commerce (未完成)	500
53.2.1 Introduction	500
54 Click Through Rate Prediction Approaches	501
54.1 Factorization Machines (FM)	502
54.1.1 Introduction	502
54.1.2 Model formulation	502
54.1.3 Model implement	505
54.1.4 Reference	519
54.2 Field-aware Factorization Machines (FFM)	520
54.2.1 Introduction	520
54.2.2 Model formulation	520
54.2.3 Model implement	522
54.2.4 References	534
54.3 Bilinear-Field-aware Factorization Machines (Bi-FFM)	535
54.3.1 Introduction	535
54.3.2 Model formulation (未完成)	536
54.4 Wide & Deep Network (WDN)	537
54.4.1 Introduction	537
54.4.2 Model formulation	537
54.4.3 Model implement	537
54.5 Deep & Cross Network (DCN)	552
54.5.1 Introduction	552
54.5.2 Model formulation	552
54.5.3 Model implement	554
54.6 DeepFM	570
54.6.1 Introduction	570
54.6.2 Model formulation	570
54.6.3 Model implement	571
54.7 xDeepFM	603
54.7.1 Introduction	603
54.7.2 Model formulation	603
54.8 Deep Interest Network (DIN)	604
54.8.1 Introduction	604
54.8.2 Model formulation	604

54.8.3 Model implement	604
54.9 Deep Interest Evolution Network (DIEN)	628
54.9.1 Introduction	628
54.9.2 Model formulation	628
54.10 AutoInt	629
54.10.1 Introduction	629
54.10.2 Model formulation	629
54.11 Convolutional Neural Networks based Click-Through Rate Prediction with Multiple Feature Sequences (未完成)	633
54.11.1 Introduction	633
54.12 DeepGBM: A Deep Learning Framework Distilled by GBDT for Online Prediction Tasks (未完成)	634
54.12.1 Introduction	634
54.13 Sequence-Aware Factorization Machines for Temporal Predictive Analytics (SeqFM)	635
54.13.1 Introduction	635
55 Deep Reinforcement Learning Approaches	636
55.1 DRN: A Deep Reinforcement Learning Framework for News Recommendation (未完成)	636
55.1.1 Introduction	636
55.2 Deep Reinforcement Learning for List-wise Recommendations (未完成)	637
55.2.1 Introduction	637
55.3 Generative Adversarial User Model for Reinforcement Learning Based Recommendation System (未完成)	638
55.3.1 Introduction	638
55.3.2 Model formulation	639
56 Generative Model Approaches	641
56.1 Collaborative Variational Autoencoder for Recommender Systems (未完成)	641
56.1.1 Introduction	641
XII Model Applications of System Security	642
57 Generative Adversarial Networks in Face Recognition & Detection	643
57.1 Adversarial Attacks on Face Detectors using Neural Net based Constrained Optimization (未完成)	643
57.1.1 Introduction	643
57.2 ADVHAT: Real-World Adversarial Attack on ARCFACE Face ID System (未完成)	644
57.2.1 Introduction	644
57.2.2 Reference	645
58 Generative Adversarial Networks in Fingerprint Recognition & Detection (未完成)	646

XIII Model Applications of Transportation	647
59 Order Dispatch	648
59.1 Large-Scale Order Dispatch in On-Demand Ride-Hailing Platforms: A Learning and Planning Approach	648
59.1.1 Introduction	648
59.1.2 Model formulation	648
XIV Numerical Optimization and Operational Research Theory and Algorithms	649
60 Optimization Overview	650
60.1 Overview	650
60.2 Reference	651
61 Unconstrained Optimization	652
61.1 Line Search Methods	652
61.1.1 General description	652
61.1.2 Choice of search direction	652
61.1.3 Choice of step length	655
61.1.4 Global convergence of line search methods	656
61.2 Trust Region Methods	657
62 Constrained Optimization	658
62.1 Interior-Point Methods for Nonlinear Programming	658
62.2 Branch and Bound Methods for Integer Programming	659
62.2.1 Reference	659
63 Robust Optimization	660
63.1 A Practical Guide to Robust Optimization	660
63.2 Distributionally Robust Optimization under Moment Uncertainty with Application to Data-Driven Problems	661
63.3 Reference	662
64 Stochastic Optimization	663
65 Operational Research Models	664
65.1 Hub Location	664
XV Mathematical Basis in Machine Learning	665
66 Mathematical Basis	666
66.1 Probability & Information Theory	666
66.1.1 Beta distribution	666

66.1.2 Wasserstein distance	667
66.2 Convex Optimization	668
66.2.1 Karush–Kuhn–Tucker Conditions (KKT)	668
66.3 Linear Algebra	669
66.3.1 Vector Gradient	669
66.3.2 Tensor Operations	670
XVI Software of Machine Learning	671
67 Deep Learning Framework	672
67.1 TensorFlow	672
67.1.1 Multi-GPUs	672
67.1.2 Multi-Workers	673
67.1.3 TensorFlow Serving	674
67.1.4 TensorFlow Estimator	675
67.1.5 Note	676
67.2 PyTorch	677
67.2.1 Multi-GPUs	677
67.2.2 Multi-Workers	678
67.3 Keras	679
67.4 Horovod	680
67.4.1 Reference	680
67.5 TensorRT	681
68 TensorFlow Project Template	682
68.1 Overall Project Architecture	682
68.1.1 Problem	684
68.1.2 Reference	684
68.2 Model	685
68.3 Engine	686
68.4 Utils	687
69 Operational Research Framework	688
69.1 Mosek	688
69.1.1 Linear programming	688
69.2 Gurobi	691
69.2.1 Mixed integer linear programming	691
69.2.2 Quadratic programming	694
69.3 Cplex	697
70 Graph Database	698
70.1 Neo4j	698

Part I

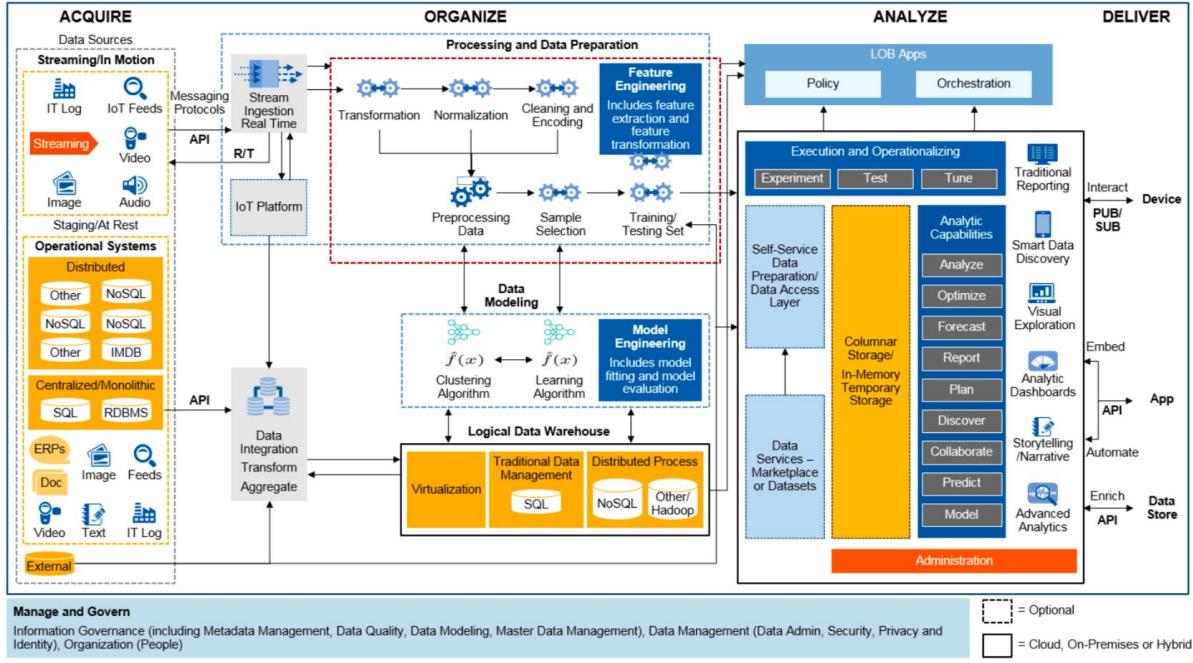
Machine Learning Basic

1 Introduction

1.1 Machine Learning System

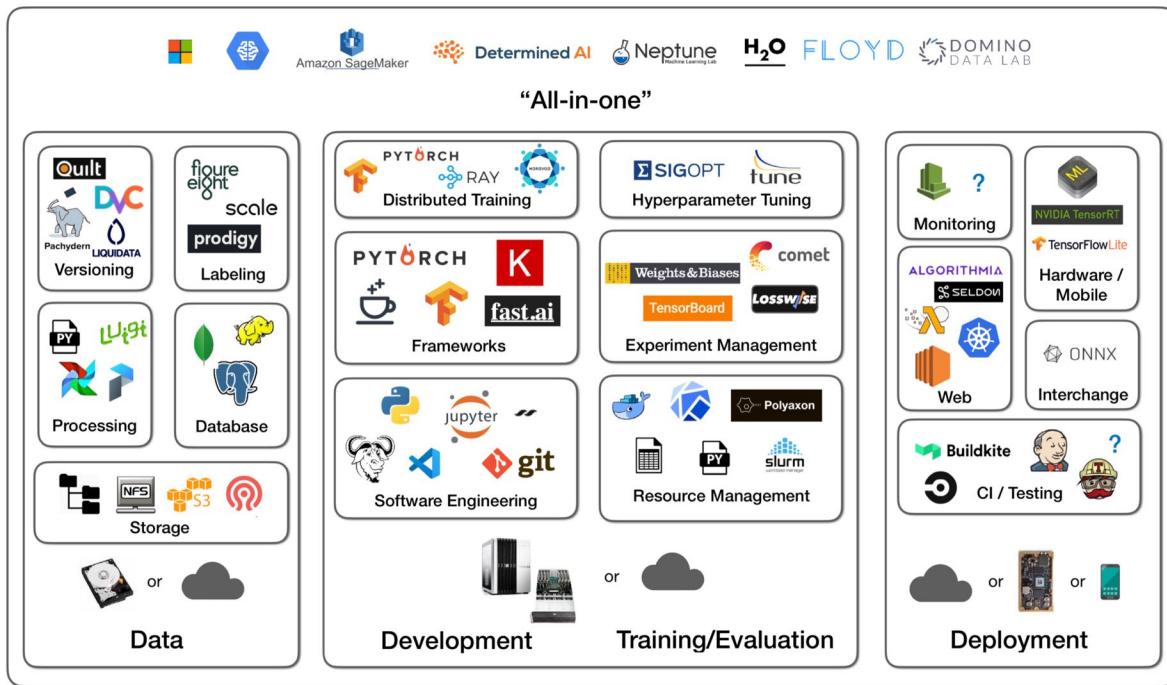
1.1.1 Architecture of End-to-End Machine learning System

完整的端到端机器学习系统平台架构如下：



- Acquisition Module:
 - API Sub-Module:
 - * ERP(Enterprise Resource Planning) Databases API
 - * IoT(Internet of things) Devices API
 - * App(Mobile application) Databases API
 - Data Ingestion Sub-Module
- Organization Module
- Analysis Module
- Delivery Module

1.1.2 Ecosystem of Machine Learning Production



Sergey Karayev at Full Stack Deep Learning Bootcamp 2019, <https://fullstackdeeplearning.com/>

图 1: Full stack of deep learning. Image Source: Production Level Deep Learning

Reference

- A Guide to Production Level Deep Learning

1.2 Framework of Machine Learning

- 完整的机器学习项目：

1. 特征框架
2. 训练框架
3. 服务框架
4. 评估框架
5. 监控框架



图 2: Megvii Brain++ AI Algorithm Platform.

- 机器学习训练“五部曲”：

1. **Dataset:** $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^N$
2. **Formulation:** $f(\mathbf{x}; \boldsymbol{\theta})$
3. **Objective:** $Obj(\boldsymbol{\theta}) = \underbrace{\mathcal{L}(\boldsymbol{\theta})}_{\text{Training loss}} + \underbrace{\Omega(\boldsymbol{\theta})}_{\text{Regularization}}$
4. **Optimization:** SGD, Adam...
5. **Evaluation:** 算法指标, 商业指标, 用户体验...

1.3 Framework of Model in Practice

1. 数据采集
2. 数据处理
3. 特征选择
4. 样本划分
5. 模型设计
6. 模型离线训练
7. 模型评估
8. 参数调优
9. 模型对比
10. 模型上线
11. 线上评估 (A/B Test)

1.4 Large-Scale Machine Learning Training System

1.4.1 Reference

- Carnegie Mellon University: Scheduling For Efficient Large-Scale Machine Learning Training

2 Dataset

2.1 Dataset Representation

本书中对数据符号 (notations) 表示作如下统一：

- N : 训练数据集的样本个数
- d : 任意样本的特征维度
- K : 离散型概率分布的类别个数
- $\mathbf{x}^{(i)} \in \mathbb{R}^d$: 第 i 个训练样本的特征向量
- $x_j^{(i)} \in \mathbb{R}$: 第 i 个训练样本的第 j 个维度
- $y^{(i)} \in \mathbb{R}$: 第 i 个训练样本的标签
- $\hat{y}^{(i)} \in \mathbb{R}$: 第 i 个训练样本的估计
- $\mathbf{y}^{(i)} \in (0, 1)^K$: 第 i 个训练样本的标签概率分布
- $\hat{\mathbf{y}}^{(i)} \in (0, 1)^K$: 第 i 个训练样本的估计概率分布
- $y_k^{(i)} \in \mathbb{R}$: 第 i 个训练样本第 k 类概率标签
- $\hat{y}_k^{(i)} \in \mathbb{R}$: 第 i 个训练样本第 k 类概率估计
- $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$: 训练数据集

对数据操作符号作如下统一：

- $\cdot \cdot$: 向量内积 (结果为一个标量)
- \odot : 向量对应位相乘 (结果为一个向量)
- \oplus : 向量的拼接 (concatenation)

2.2 Independent Variable

2.2.1 Category features

One-hot encoding

- 类别型属性本质是字符串
- 只有数值才能被计算机（算法）识别
- One-hot 编码方式将每一个 field 的字符串分类 \Rightarrow 离散型 field 值概率分布

原始数据：

性别	国籍
男	中国
女	美国
男	新加坡

独热编码之后的数据：

性别 = 男	性别 = 女	国籍 = 中国	国籍 = 美国	国籍 = 新加坡
1	0	1	0	0
0	1	0	1	0
1	0	0	0	1

2.2.2 Numerical features

Discretization

问题：Why discretizing a numerical feature into categorical features ? 在工业界，很少直接将连续型特征 (numerical features) 直接作为 Logistic Regression 模型的输入特征，而是将某一连续的特征离散化为一系列 0/1 特征再交给 Logistic Regression 模型，优势如下：

- 对异常数据有强鲁棒性。例如：age=300
- 模型更加稳定。例如：对用户的年龄进行离散化，20-30 作为一个区间，不会因为一个用户的年龄长了一岁就变成一个完全不同的人。当然处于区间相邻处的样本刚好相反，所以如何划分区间是门学问，一般有几种：
 - 专家经验
 - 数据分位点
 - 信息增益最大化
- 引入非线性变换。逻辑回归属于广义的线性模型，因此表达能力受到了限制，某一连续型特征离散化为 N 个 0-1 特征后，每个 0-1 特征有单独的权重，相当于为模型引入了非线性，能够提升模型的表达能力，加大拟合程度。
- 便于之后进行特征交叉。离散化后可以进行特征交叉 (feature interactions)，进一步引入非线性，提升模型表达能力。
- 降低了模型过拟合的风险。
- 离散特征的增加和减少都很容易，易于模型的快速迭代。
- 稀疏向量内积乘法速度快，计算结果方便存储，容易扩展。

Discretization algorithms 参考综述性论文 A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning [?]

2.3 Dependent Variable

2.3.1 Imbalanced dataset in classification task

1. 正样本重采样 (Oversampling)

2. 负样本下采样 (Undersampling)

3. 正样本上的 SMOTE[?] 算法

Oversampling 以 Logistic Regression 为例，说明 oversample 对模型的影响：

$$\begin{aligned} z &= \sum_{j=1}^d w_j x_j + b \\ \phi(z) &= \frac{1}{1 + \exp(-z)} \\ \therefore \phi(\mathbf{x}; \mathbf{w}, b) &= \frac{1}{1 + \exp(-\sum_{i=1}^d w_i x_i - b)} \end{aligned}$$

loss function 如下：

$$J(\mathbf{w}, b) = \underbrace{-\frac{1}{N} \sum_{i=1}^N (y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))}_{\text{training loss}} + \underbrace{\frac{\lambda}{2} \mathbf{w}^T \mathbf{w}}_{\text{l2 regularization}}$$

因此模型系数 (slope) 和截距 (intercept) 的负梯度如下：

$$\begin{aligned} -\frac{\partial J(\mathbf{w}, b; \mathbf{x}^{(i)}, y^{(i)})}{\partial w_j} &= \frac{\partial}{\partial w_j} \left(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right) - \frac{\partial}{\partial w_j} \left(\frac{\lambda}{2} \mathbf{w}^T \mathbf{w} \right) \\ &= \frac{y^{(i)}}{\hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial w_j} + \frac{1 - y^{(i)}}{1 - \hat{y}^{(i)}} (-1) \frac{\partial \hat{y}^{(i)}}{\partial w_j} - \lambda w_j \\ &= \frac{\hat{y}^{(i)}(y^{(i)} - 1) + y^{(i)}(1 - \hat{y}^{(i)})}{\hat{y}^{(i)}(1 - \hat{y}^{(i)})} \frac{\partial \hat{y}^{(i)}}{\partial w_j} - \lambda w_j \\ \therefore \frac{\partial \hat{y}^{(i)}}{\partial w_j} &= \frac{\partial \phi(\mathbf{x}^{(i)}; \mathbf{w}, b)}{\partial w_j} = \frac{\partial \phi(z)}{\partial z} \frac{\partial z}{\partial w_j} \\ &= (1 - \phi(z)) \phi(z) x_j^{(i)} = (1 - \hat{y}^{(i)}) \hat{y}^{(i)} x_j^{(i)} \\ \therefore -\frac{\partial J(\mathbf{w}, b; \mathbf{x}^{(i)}, y^{(i)})}{\partial w_j} &= \frac{\hat{y}^{(i)} y^{(i)} - \hat{y}^{(i)} + y^{(i)} - \hat{y}^{(i)} y^{(i)}}{\hat{y}^{(i)}(1 - \hat{y}^{(i)})} (1 - \hat{y}^{(i)}) \hat{y}^{(i)} x_j^{(i)} - \lambda w_j \\ &= (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)} - \lambda w_j \\ -\frac{\partial J(\mathbf{w}, b; \mathbf{x}^{(i)}, y^{(i)})}{\partial b} &= \frac{\partial}{\partial b} \left(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right) - \underbrace{\frac{\partial}{\partial b} \left(\frac{\lambda}{2} \mathbf{w}^T \mathbf{w} \right)}_{=0} \\ &= \frac{y^{(i)}}{\hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial b} + \frac{1 - y^{(i)}}{1 - \hat{y}^{(i)}} (-1) \frac{\partial \hat{y}^{(i)}}{\partial b} \\ &= \frac{\hat{y}^{(i)}(y^{(i)} - 1) + y^{(i)}(1 - \hat{y}^{(i)})}{\hat{y}^{(i)}(1 - \hat{y}^{(i)})} \frac{\partial \hat{y}^{(i)}}{\partial b} \end{aligned}$$

$$\begin{aligned}
\therefore \frac{\partial \hat{y}^{(i)}}{\partial b} &= \frac{\partial \phi(\mathbf{x}^{(i)}; \mathbf{w}, b)}{\partial b} = \frac{\partial \phi(z)}{\partial z} \frac{\partial z}{\partial b} \\
&= (1 - \phi(z)) \phi(z) \cdot 1 = (1 - \hat{y}^{(i)}) \hat{y}^{(i)} \\
\therefore -\frac{\partial J(\mathbf{w}, b; \mathbf{x}^{(i)}, y^{(i)})}{\partial b} &= \frac{\hat{y}^{(i)} y^{(i)} - \hat{y}^{(i)} + y^{(i)} - \hat{y}^{(i)} y^{(i)}}{\hat{y}^{(i)}(1 - \hat{y}^{(i)})} (1 - \hat{y}^{(i)}) \hat{y}^{(i)} \\
&= y^{(i)} - \hat{y}^{(i)}
\end{aligned}$$

因此（以 SGD 为例）参数的更新公式为：

$$\begin{aligned}
w_j &\leftarrow w_j + \eta \left[(y^{(i)} - \hat{y}^{(i)}) x_j^{(i)} - \lambda w_j \right] \\
b &\leftarrow b + \eta (y^{(i)} - \hat{y}^{(i)})
\end{aligned}$$

oversampling 对模型的影响总结如下：

- **Oversampling 不影响模型的系数 (slope)，但是会放大模型的截距 (intercept)**（因为当 oversample 正样本时，正样本数量增多，参数 b 的更新会使得其数值越来越大 $\because \hat{y}^{(i)} < 1$ ）
- 因为截距放大了，**预测事件的概率也都变大了**
- 评价准则中，Sensitivity = $\frac{\text{TP}}{\text{TP}+\text{FN}}$ （真实为正，预测为正）和 Specificity = $\frac{\text{TN}}{\text{TN}+\text{FP}}$ （真实为负，预测为负）不受影响，但是 False Positive Rate FPR = $\frac{\text{FP}}{\text{FP}+\text{TN}}$ （真实为负，预测为正）和 False Negative Rate FNR = $\frac{\text{FN}}{\text{FN}+\text{TP}}$ （真实为正，预测为负）会受影响
- ROC curve 不受影响

oversampling 之后如何修正？

- 假设： P_0 为对某个样本预测为 0（负类）的概率（在 oversample 后的训练样本上学习后）， $P_1 = 1 - P_0$ 为预测为 1（正类）的概率
- 令： $A = \frac{P_1}{\text{Oversample}_{\text{后target 为 1}} \text{ 的训练样本比例}} / \frac{P_0}{\text{Oversample}_{\text{后target 为 0}} \text{ 的训练样本比例}}$ ， $B = \frac{P_0}{\text{Oversample}_{\text{后target 为 0}} \text{ 的训练样本比例}} / \frac{P_1}{\text{Oversample}_{\text{后target 为 1}} \text{ 的训练样本比例}}$
- 修正后的概率为： $P_1 = \frac{A}{A+B}$ ， $P_0 = \frac{B}{A+B}$

SMOTE 算法具体做法如下：

1. 找到一个正样本 x_1 (少数类别的样本)
2. 根据 KNN 计算找出 x_i 最近邻的 k 个正样本 (该例中 k=5, 找到的近邻正样本为 x_2, x_3, x_4, x_5, x_6)
3. 在 x_1 和选出的近邻正样本的线段上生成新的正样本 (凸组合), 作为新的正样本, 对原始数据集进行扩充

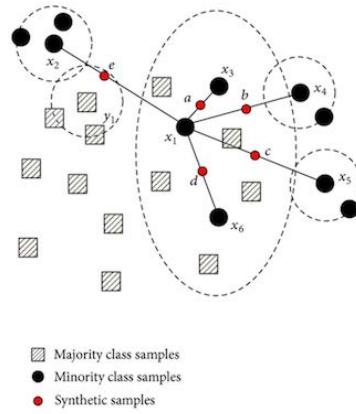


图 3: Visualization of SMOTE

优劣对比

- 正样本上的 Oversampling 从很小的正样本池中引入重复样本, 因此容易导致模型过拟合;
- 负样本上的 Undersampling 可能会漏掉在两个类中提供关键差异信息的重要样本, 因此容易导致模型效果不佳;
- SMOTE 利用正样本的信息合成了新的正样本, 而不是重复已有的正样本, 因此在一定程度上缓解了过拟合, 但是这依旧不能缓解所有的过拟合问题, 因为它的生成还是来自于已有的正样本。

2.4 Dataset Preprocessing

2.4.1 Image preprocessing

2.4.2 Text preprocessing

NLP 模型任务（以英文任务为例）基本工作流程如下：

1. 文本获取 (Raw text data)
2. 分词 (Segmentation)
3. 清洗 (Cleaning)
 - 去除标点符号
 - 英文大写转换为小写
 - 数字归一化
 - 停用词库 & 低频词库
 - ...
4. 标准化 (Normalization)
 - 词形还原 (Lemmatization) (转变 e.g. driving \Rightarrow drive, drove \Rightarrow drive)
 - 词干提取 (Stemming) (缩减 e.g. cats \Rightarrow cat, effective \Rightarrow effect)
5. 特征提取 (Feature extraction)
 - TF-IDF
 - Word2Vec
 - CountVectorizer
6. 建模 (Modeling)

2.5 Analysis of Dataset (未完成)

2.5.1 Significance Analysis

2.5.2 Correlation Analysis

2.5.3 Principal Component Analysis (PCA)

主成分分析 (Principal Component Analysis, PCA) 是一种常用的无监督数据降维方法，使得原始高维空间 \mathcal{X} 内的高维特征向量集 $\mathcal{D}_{\mathcal{X}} = \{\mathbf{x}^{(i)} \in \mathbb{R}^{\mathcal{X}}\}_{i=1}^N$ 在转换 $\mathcal{X} \rightarrow \mathcal{Z}$, $|\mathcal{X}| > |\mathcal{Z}|$ 后的低维空间 \mathcal{Z} 中低维特征向量集 $\mathcal{D}_{\mathcal{Z}} = \{\mathbf{z}^{(i)} \in \mathbb{R}^{\mathcal{Z}}\}_{i=1}^N$ 的方差最大。

Methodology 原始高维空间 \mathcal{X} 内样本集的均值 (中心点) 为：

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}^{(i)}$$

投影空间 \mathcal{Z} 内所有样本的方差为：

$$\begin{aligned}\sigma(\mathcal{D}_{\mathcal{Z}}) &= \frac{1}{N} \sum_{i=1}^N (\mathbf{z}^{(i)} - \bar{\mathbf{z}})^2 \\ &= \frac{1}{N} \sum_{i=1}^N (\mathbf{w} \cdot \mathbf{x}^{(i)} - \mathbf{w} \cdot \bar{\mathbf{x}})^2 \quad (\text{考虑特例：投影空间 } \mathcal{Z} \text{ 为一维空间，} z = \mathbf{w} \cdot \mathbf{x})\end{aligned}$$

2.5.4 Linear Discriminant Analysis (LDA)

2.6 Summary of Dataset

- 模型对数据的使用，其实是两种模式的权衡：
 - 海量离散特征 + 简单模型 (Logistic Regression)
 - 少量连续特征 + 复杂模型 (XGBoost)

也就是离散化的特征加线性模型，或者数值型特征加深度学习。二者的选择就是看喜欢折腾特征还是折腾模型了。通常而言，前者容易，而且可以 n 个人一起并行做，有成功经验，后者目前看着很赞，但是能走多远还拭目以待。

- **数据决定了结果的上限（天花板），而模型只决定距离这个“天花板”还有多远。**

3 Objective Function

3.1 Expected risk & Empirical risk

3.1.1 Expected risk minimization

期望风险 (Expected risk) 是对输入 (input: $x \in \mathbb{R}^{\mathcal{X}}$) 和输出 (output: $y \in \mathbb{R}^{\mathcal{Y}}$) 联合分布空间 $\mathbb{R}^{\mathcal{X}} \times \mathbb{R}^{\mathcal{Y}}$ 上对目标函数的度量，记联合概率分布 (joint probability distribution) 为：

$$\mathcal{P} : \mathbb{R}^{\mathcal{X}} \times \mathbb{R}^{\mathcal{Y}} \rightarrow \mathbb{P}$$

那么期望风险目标函数 $R(h) : \mathbb{R}^{\mathcal{X}} \times \mathbb{R}^{\mathcal{Y}} \rightarrow \mathbb{R}$ 为：

$$\begin{aligned} R(h_{\theta}) &= \mathbb{E} [\mathcal{L}(h_{\theta}(x), y)] \\ &= \int_{\mathbb{R}^{\mathcal{X}} \times \mathbb{R}^{\mathcal{Y}}} \mathcal{L}(h_{\theta}(x), y) d\mathcal{P}(x, y) \end{aligned}$$

其中 θ 为 hypothesis function: $h : \mathcal{X} \rightarrow \mathcal{Y}$ 的参数，而最终得到的最优 hypothesis function h^* 为全部假设空间 \mathcal{H} 内，使得期望风险 $R(h)$ 最小的那个：

$$h^* = \arg \min_{h \in \mathcal{H}} R(h)$$

3.1.2 Empirical risk minimization

通常，由于分布 $\mathcal{P}(x, y)$ 未知，所以导致期望风险 $R(h)$ 不能够被直接计算，进而导致最优假设函数 h^* 不可得。然而我们可以通过对训练集 $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$ 上的损失函数取平均值来计算近似值，称为经验风险 (Empirical risk) $R_{\text{emp}}(h) : \mathbb{R}^{\mathcal{X}} \times \mathbb{R}^{\mathcal{Y}} \rightarrow \mathbb{R}$ ：

$$R_{\text{emp}}(h_{\theta}) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(h_{\theta}(x^{(i)}), y^{(i)})$$

其中训练集 \mathcal{D} 中的所有样本来自于：**独立同分布假设下，在样本联合分布空间 $\mathbb{R}^{\mathcal{X}} \times \mathbb{R}^{\mathcal{Y}}$ 内，基于联合概率分布 $\mathcal{P}(x, y)$ 的 n 次有放回采样**。则最优的假设函数 h^* 为：

$$h^* = \arg \min_{h \in \mathcal{H}} R_{\text{emp}}(h)$$

3.2 Loss Function for Regression

- input feature vector: $x \in \mathbb{R}^X$
- label of output scalar: $y \in \mathbb{R}$
- hypothesis function(output a predicted real-value scalar): $h_\theta : \mathbb{R}^X \rightarrow \mathbb{R}$

3.2.1 Mean squared error

$$\min_{\theta} J(\theta) = \frac{1}{N} \sum_{i=1}^N (y^{(i)} - h_\theta(x^{(i)}))^2$$

3.2.2 Mean absolute error

$$\min_{\theta} J(\theta) = \frac{1}{N} \sum_{i=1}^N |y^{(i)} - h_\theta(x^{(i)})|$$

3.2.3 Huber loss

Definition of **Huber loss** is as follow:

$$\mathcal{L}_\delta(y, h_\theta(x)) = \begin{cases} \frac{1}{2}(y - h_\theta(x))^2 & \text{if } |y - h_\theta(x)| \leq \delta, \\ \delta |y - h_\theta(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

and the empirical risk with huber loss is as follow:

$$\min_{\theta} J(\theta) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}_\delta(y^{(i)}, h_\theta(x^{(i)}))$$

Comparison among Squared loss, Absolute loss and Huber loss:

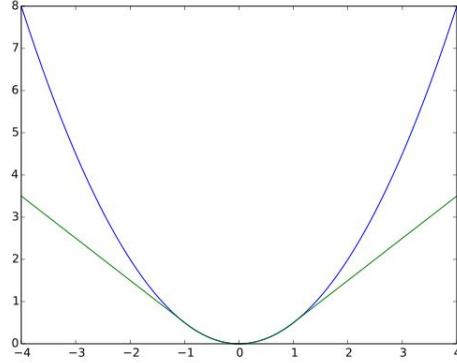


图 4: Huber loss ($\delta = 1$) and squared error loss (blue) as a function of $y - h_\theta(x)$.

- Squared loss:

- 优点：最常用的损失函数，对非异常点情况下，存在较大误差的点有较好的关注。
- 缺点：**会对异常点 (outlier) (x, y) 施以较大的惩罚**，即当异常点很多时，目标函数过于关注对异常点的惩罚，从而导致模型 h_θ 会出现比较大偏差，因此**不够鲁棒 (robust)**。

- Absolute loss:

- 优点：当存在较多异常点 (outlier) 时，MAE 的表现较好。
- 缺点：在 $y - h_\theta(x) = 0$ 处时不可导（虽然一般优化任务中不存在训练残差为 0 的情况）。

- Huber loss:

- 优点：比较**Robust (对异常值不敏感)** 的损失函数，同时在 $y - h_\theta(x) = 0$ 处可导。

3.3 Loss Function for Classification

3.3.1 Hinge loss

Definition of **Hinge loss** is as follow:

$$\mathcal{L}(y, h_\theta(x)) = \max(0, 1 - y \cdot h_\theta(x))$$

so:

$$\min_{\theta} J(\theta) = \frac{1}{N} \sum_{i=1}^N \max(0, 1 - y^{(i)} \cdot h_\theta(x^{(i)}))$$

where :

- $y \in \{-1, +1\}$ is label output
- $h_\theta(x) \in (-1, +1)$ is predicted output

含义：

- 正确分类时， y 与 $h_\theta(x)$ 同号，此时的损失值域 $\mathcal{L} \in [0, 1)$
- 错误分类时， y 与 $h_\theta(x)$ 异号，此时的损失值域 $\mathcal{L} \in [0, +\infty)$
- 考虑预测类别和真实类别相同（预测正确）的情况：
 - $y = +1$ 且 $h_\theta(x) = +100000$ 时，预测类别和真实类别相同
 - 且此时 $1 - y \cdot h_\theta(x) = 1 - 1 \times 100000 = -99999$ ，即理论上应该得到的奖励为 $-(1 - y \cdot h_\theta(x)) = 99999$
 - 但是此时 loss 为 $\mathcal{L} = \max(0, 1 - y \cdot h_\theta(x)) = 0$ ，即实际上得到的奖励为 $-\mathcal{L} = 0$ ，即此时不会得到任何奖励。

因此，**Hinge loss 不鼓励模型 h_θ 过分自信**，即预测输出 $h_\theta(x) \in (-1, +1)$ 就可以了，不用 $|h_\theta(x)| > 1$ 。

- 考虑预测类别和真实类别不同（预测错误）的情况：
 - $y = +1$ 且 $h_\theta(x) = -100000$ 时，预测类别和真实类别相反
 - 且此时 $1 - y \cdot h_\theta(x) = 1 - 1 \times (-100000) = 100001$ ，即理论上应该得到的奖励为 $-(1 - y \cdot h_\theta(x)) = -100001$
 - 此时 loss 为 $\mathcal{L} = \max(0, 1 - y \cdot h_\theta(x)) = 100001$ ，即实际上得到的奖励为 $-\mathcal{L} = -100001$ ，惩罚很大，因此需要着重针对该样本进行参数学习，努力将其分类正确。

因此，Hinge loss 更加关注分类错误样本，而不太关注分类正确样本的学习，所以可能导致的效果是**maximum-margin**，即：

- 正样本和负样本之间的 $h_\theta(x)$ 差异非常明显
- 正样本和正样本之间的 $h_\theta(x)$ 差异不明显
- 负样本和负样本之间的 $h_\theta(x)$ 差异不明显

3.3.2 Log-likelihood

- input feature vector: $x \in \mathbb{R}^X$
- label of output: $y \in \mathcal{Y} = \{1, \dots, K\}$, K is total number of classes.
- label of output probability distribution: $p \in \mathbb{P}^{\mathcal{Y}}$
- hypothesis function(output a predicted probability distribution over all classes): $h_{\theta} : \mathbb{R}^X \rightarrow \mathbb{P}^{\mathcal{Y}}$

基于样本独立同分布的假设，最大化所有用于训练的观测样本 $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$ 发生的概率，即可通过训练得到样本输入特征向量 x 与输出标签概率分布 y 之间的关系（近似真实）：

$$\begin{aligned}
 & \max P(\mathcal{D}) \\
 \Rightarrow & \max \prod_{i=1}^N p(x^{(i)}, y^{(i)}) \quad (\because \text{i.i.d}) \\
 \Rightarrow & \max \prod_{i=1}^N p(y^{(i)}|x^{(i)})p(x^{(i)}) \quad (\because p(x, y) = p(y|x)p(x)) \\
 \Rightarrow & \max \frac{1}{N} \prod_{i=1}^N p(y^{(i)}|x^{(i)}) \quad (\because p(x^{(i)}) \approx \frac{1}{N}) \\
 \Rightarrow & \max \frac{1}{N} \log \left(\prod_{i=1}^N p(y^{(i)}|x^{(i)}) \right) \\
 \Rightarrow & \max \frac{1}{N} \sum_{i=1}^N \log p(y^{(i)}|x^{(i)})
 \end{aligned}$$

3.3.3 Kullback–Leibler divergence

Convert maximizing Log-likelihood into minimizing KL-divergence:

$$\begin{aligned}
 & \max \frac{1}{N} \sum_{i=1}^N \log p(y^{(i)}|x^{(i)}) \\
 \Rightarrow & \min \frac{1}{N} \sum_{i=1}^N D_{\text{KL}}(p^{(i)} \| h_{\theta}(x^{(i)})) \quad (\text{i.e. } y^{(i)} \sim p^{(i)} \in \mathbb{P}^{[K]}) \\
 \Rightarrow & \min \frac{1}{N} \sum_{i=1}^N D_{\text{KL}}(p^{(i)} \| q^{(i)}) \quad (\text{i.e. } q^{(i)} = h_{\theta}(x^{(i)}) \in \mathbb{P}^{\mathcal{Y}}) \\
 \Rightarrow & \min \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log \frac{p_k^{(i)}}{q_k^{(i)}} \quad (\text{definition of Kullback–Leibler divergence}) \\
 \Rightarrow & \min \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K \left(p_k^{(i)} \log p_k^{(i)} - p_k^{(i)} \log q_k^{(i)} \right) \\
 \Rightarrow & \underbrace{\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log p_k^{(i)}}_{\text{entropy}} - \underbrace{\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log q_k^{(i)}}_{\text{cross entropy}}
 \end{aligned}$$

$$\Rightarrow \min_{\theta} J(\theta) = \underbrace{\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log p_k^{(i)}}_{\text{constant}} - \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log \left(h_{\theta}(x^{(i)}) \right)_k$$

Step-by-step understanding of KL-divergence:

- 信息量 (Information) :

$$I = -\log p, \quad p \in \mathbb{P}$$

表示一个概率事件的“震惊”程度。 概率越小，信息量越大 (e.g. 国足世界杯夺冠)。



图 5: 国足世界杯夺冠的概率 p 很小，因此信息量 $I = -\log p$ 很大

- 熵 (Entropy) :

$$\mathcal{H}(p) = \sum_{k=1}^K p_k (-\log p_k) = -\sum_{k=1}^K p_k \log p_k$$

表示一个概率分布 p 的期望信息量。

- KL 散度 (KL-divergence) :

$$\begin{aligned} D_{\text{KL}}(p \parallel q) &= \sum_{k=1}^K p_k \log \frac{p_k}{q_k} \\ &= \sum_{k=1}^K (p_k \log p_k - p_k \log q_k) \\ &= \sum_{k=1}^K \left(p_k (-\log q_k) - p_k (-\log p_k) \right) \end{aligned}$$

表示两个概率分布之间的差异。

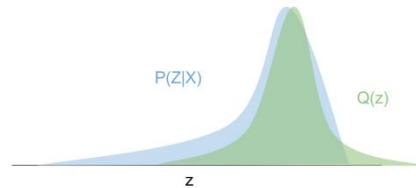


图 6: KL-Divergence 用于衡量两个概率分布之间的差异

KL-Divergence 中**含有训练参数 θ 的部分得在 q ，常数部分在 p** ，因为 KL 散度没有距离定义上的对称性，所以反过来无法优化计算。

3.3.4 Cross entropy

Definition of **Cross entropy** between the distributions \mathbf{p} and \mathbf{q} is as follow:

$$\mathcal{H}(\mathbf{p}, \mathbf{q}) = \mathbb{E}_p [-\log q] = - \sum_{k=1}^K p_k \log q_k = - \langle \mathbf{p}, \log \mathbf{q} \rangle$$

where the **inner product** $\langle \cdot, \cdot \rangle$ computes a similarity measure between the network prediction \mathbf{q} and the corresponding data label \mathbf{p} , and:

- $\mathbf{p} \in \mathbb{P}^{[K]}$: target probability distribution(**always 0-1 probability distribution**) over K classes
- $\mathbf{q} = h_\theta(x) \in \mathbb{P}^{[K]}$: predicted probability distribution over K classes:

$$h_\theta(x) = [\mathbb{P}(\text{class}(x) = 1), \quad \mathbb{P}(\text{class}(x) = 2), \quad \dots, \quad \mathbb{P}(\text{class}(x) = K)].$$

Directly drop entropy term in KL-divergence, we can get empirical risk with cross entropy loss as follow:

$$\begin{aligned} & \min \underbrace{\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log p_k^{(i)}}_{\text{entropy}} - \underbrace{\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log q_k^{(i)}}_{\text{cross entropy}} \\ \Rightarrow & \min_{\theta} J(\theta) = - \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log q_k^{(i)} \\ \Rightarrow & \min_{\theta} J(\theta) = - \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log [h_\theta(x^{(i)})]_k \\ \Rightarrow & \min_{\theta} J(\theta) = - \frac{1}{N} \sum_{i=1}^N \left\langle p^{(i)}, \log h_\theta(x^{(i)}) \right\rangle \quad (\text{inner product operation between vectors}) \end{aligned}$$

3.3.5 Focal loss

Cross Entropy We introduce the **focal loss**[?] starting from the cross entropy (CE) loss for binary classification:

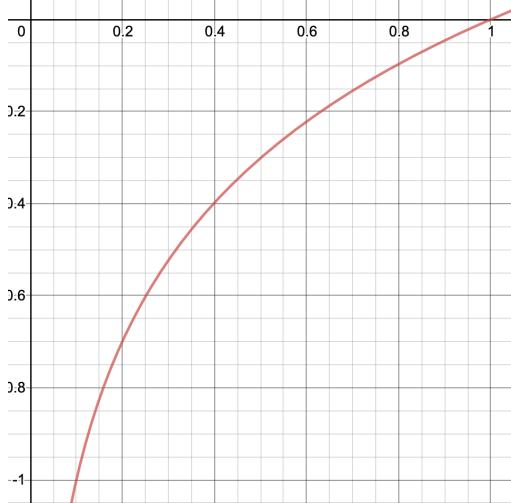
$$\begin{aligned} \text{CE}(p, y) &= -\left(y \log(p) + (1-y) \log(1-p)\right) \\ &= \begin{cases} -\log(p) & \text{if } y = 1, \\ -\log(1-p) & \text{otherwise.} \end{cases} \end{aligned}$$

where $y \in \{0, 1\}$ specifies the ground-truth class and $p \in [0, 1]$ is the model's estimated probability for the class with label $y = 1$. For notational convenience, define the **probability of classification correctly**:

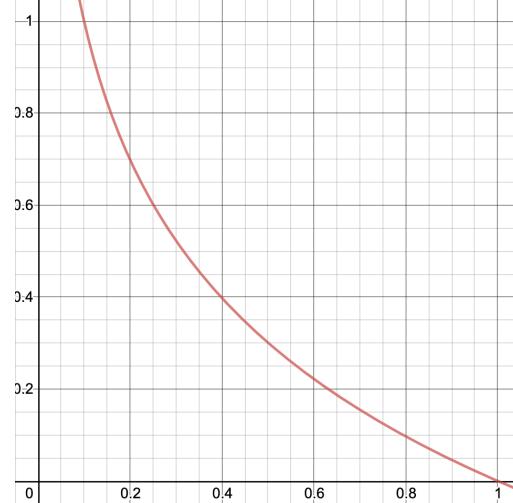
$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise.} \end{cases}$$

we can rewrite cross entropy(even for multi classification task) as follow:

$$\text{CE}(p, y) = \text{CE}(p_t) = -\log(p_t)$$



(a) $\log(p_t)$ as a reward of correct classification.



(b) $-\log(p_t)$ as a loss of correct classification.

CE loss 存在的一个关键问题是：样本集合几乎为易分类样本 ($p_t \gg 0.5$) 且非平衡样本 ($n_{\text{pos}} \ll n_{\text{neg}}$) 时，训练阶段对正样本的学习效果很差，进而导致预测阶段对正样本的预测准确率很低。

- 假定所有的样本：

$$\mathcal{D} = \left\{ (x^{(i)}, y^{(i)}) | x^{(i)} \in \mathbb{R}^{\mathcal{X}}, y^{(i)} \in \{0, 1\} \right\}_{i=1}^N$$

都是易分类样本：

$$p_t \gg 0.5$$

- 那么当 p_{ture} 来自于一个相对较高的置信区间时 (e.g. $p_t \in (0.7, 0.98)$)，样本集合 \mathcal{D} 上的总 CE loss 均来自于易分类样本带来的不大也不小的损失 $\text{CE}(p_t) = -\log(p_t)$

- 但是对于易分类样本，我们并不知道其样本是正样本 $y^{(i)} = 1$ 还是负样本 $y^{(i)} = 0$ ，因此在**不平衡样本**情况下，样本集合 \mathcal{D} 上的总 CE loss 几乎来自于负样本：

$$\begin{aligned} \sum_{i=1}^N -\log(p_t^{(i)}) &= \sum_{i=1}^{n_{\text{neg}}} -\log(p_t^{(i)}) + \sum_{i=1}^{n_{\text{pos}}} -\log(p_t^{(i)}) \quad (n_{\text{neg}} + n_{\text{pos}} = n) \\ &\approx \sum_{i=1}^{n_{\text{neg}}} -\log(p_t^{(i)}) \quad (\because n_{\text{pos}} \ll n_{\text{neg}}) \end{aligned}$$

因此，这就造成了将正样本（少数类别样本）的 CE loss 淹没在大海中，进而导致：

- 训练阶段：对负样本的学习严重冗余，而对**正样本的学习严重匮乏**
- 测试阶段：对负样本的预测准确率很高，而对**正样本的预测准确率很低**

Balanced Cross Entropy 因此为了平衡正负样本的 loss，引入了一个权重因子（weight factor）： $\alpha \in [0, 1]$ ，即：

$$\sum_{i=1}^N -\log(p_t^{(i)}) = \sum_{i=1}^{N_{\text{neg}}} -(1-\alpha) \log(p_t^{(i)}) + \sum_{i=1}^{N_{\text{pos}}} -\alpha \log(p_t^{(i)})$$

为了简化表示，可以记为：

$$\alpha_t = \begin{cases} \alpha \gg 0.5 & \text{if } y = 1 \text{ (positive sample),} \\ 1 - \alpha \ll 0.5 & \text{otherwise (negative sample).} \end{cases}$$

因此之前单一样本上的 CE loss:

$$\text{CE}(p_t) = -\log(p_t)$$

经过**考虑正负样本的加权**后变为：

$$\text{CE}(p_t) = -\alpha_t \log(p_t)$$

Note: How to set α_t ?

- inverse class frequency

$$\alpha_t = \begin{cases} \frac{1}{N_{\text{pos}}} & \text{if } y = 1 \text{ (positive sample),} \\ \frac{1}{N_{\text{neg}}} & \text{otherwise (negative sample).} \end{cases}$$

- reciprocal

$$\alpha_t = \begin{cases} \frac{N_{\text{neg}}}{N} & \text{if } y = 1 \text{ (positive sample),} \\ \frac{N_{\text{pos}}}{N} & \text{otherwise (negative sample).} \end{cases}$$

- hyperparameter to set by cross validation

Focal Loss Definition 很多任务（e.g. Dense Object Detection）的数据存在以下特点：

- 容易判别分类 ($p_t \gg 0.5$) 的负样本 ($y = 0$) 占据了总 loss 的大部分，并主导了梯度
- 虽然 α_t 平衡了正负样本的重要性（balances the importance of positive/negative examples），但是其并没有让损失函数可以有效辨别划分难易样本（does not differentiate between easy/hard examples），进而更加关注难分的样本（hard example）

因此针对难分的样本，设置损失函数如下：

$$\text{FL}(p_t) = -(1 - p_t)^\gamma \log(p_t)$$

其中 $\gamma \geq 0$ 为可调的超参数。该公式解释如下：

- 当样本很难分时：
 - $p_t \rightarrow 0$ 很小
 - 该样本本身的损失为 $\text{CE}(p_t) = -\log(p_t)$ 大
 - 而因为难分，我们希望更加关注它，即给该样本设置更大的难易重要度 (importance)，因此 $(1 - p_t)^\gamma \rightarrow 1$ 很大
- 当样本容易分时：
 - $p_t \rightarrow 1$ 很大
 - 该样本本身的损失为 $\text{CE}(p_t) = -\log(p_t)$ 小
 - 而因为易分，我们希望相对更忽略它，即给该样本设置更小的难易重要度 (importance)，因此 $(1 - p_t)^\gamma \rightarrow 0$ 很小

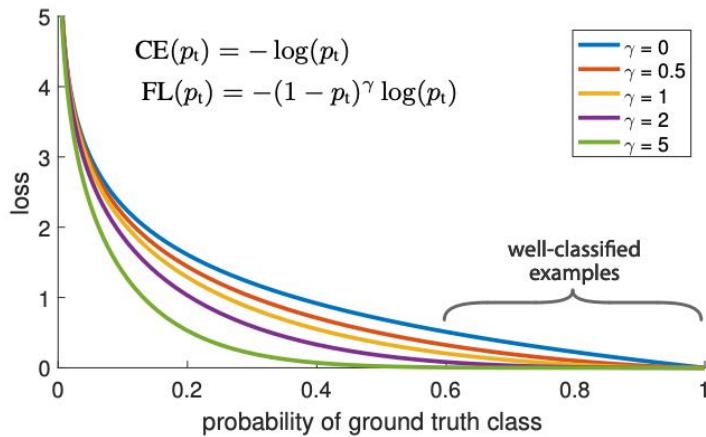


图 7: We propose a novel loss we term the Focal Loss that adds a factor $(1 - p_t)^\gamma$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > 0.5$), putting more focus on hard, misclassified examples.

Focal loss 相对于 CE loss 的优点是：在样本普遍易分类 ($p_t > 0.5$) 的情况下，更加关注那些相对模棱两可 (e.g. $p_t = 0.55$) 的样本，赋予其指数级别相对高的重要性权重。

考虑 Balanced Cross Entropy，因此最终 Focal Loss 为：

$$\text{FL}(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t)$$

训练样本上的 Focal Loss 为：

$$\min_{\theta} J(\theta) = \frac{1}{N} \sum_{i=1}^N -\alpha_t(1 - p_t^{(i)})^\gamma \log(p_t^{(i)})$$

因此 Focal Loss 的适用场景是：**训练样本几乎均为易分类样本，且正负样本不平衡，即：普遍为相对易分类 ($p_t > 0.5$) 的负样本 ($y = 0$)。** Focal loss 相对于 CE loss 的缺点是：极度不稳定，因此不建议直接适用 Focal Loss，最好是经过几个 epoch 之后再将 Cross Entropy 改为 Focal Loss，或者只是将 Focal Loss 作为 Fine-tune。

3.3.6 Gradient Harmonized Mechanism (GHM)

内容来自论文Gradient Harmonized Single-stage Detector[?]，该方法改进了Focal Loss[?] 中超参数 (α, γ) 需要耗费大量工作进行调参的难题。

Problem Description Consider the basic binary cross entropy loss :

$$\mathcal{L}_{\text{CE}}(p^*, p_\theta(x)) = \begin{cases} -\log(p_\theta(x)) & \text{if } p^* = 1 \\ -\log(1 - p_\theta(x)) & \text{if } p^* = 0 \end{cases}$$

where:

- $p_\theta(x) \in [0, 1]$: probability predicted by the model
- $p^* \in \{0, 1\}$: ground-truth label

Note $z_\theta(x) \in \mathbb{R}$ as the logit output of model, therefore :

$$p_\theta(x) = \sigma(z_\theta(x)) \quad (\sigma(\cdot) \text{ means the sigmoid function.})$$

gradient of binary cross entropy loss with regard to $z_\theta(x)$:

$$\begin{aligned} \frac{\partial \mathcal{L}_{\text{CE}}(p^*, p_\theta(x))}{\partial z_\theta(x)} &= \frac{\partial \mathcal{L}_{\text{CE}}(p^*, p_\theta(x))}{\partial p_\theta(x)} \frac{\partial p_\theta(x)}{\partial z_\theta(x)} \\ &\because \frac{\partial p_\theta(x)}{\partial z_\theta(x)} = \frac{\partial}{\partial z_\theta(x)} \left(\frac{1}{1 + e^{-z_\theta(x)}} \right) \\ &= (1 - \sigma(z_\theta(x)))\sigma(z_\theta(x)) \\ &= (1 - p_\theta(x)) \cdot p_\theta(x) \\ &\therefore \frac{\partial \mathcal{L}_{\text{CE}}(p^*, p_\theta(x))}{\partial p_\theta(x)} = \begin{cases} -\frac{1}{p_\theta(x)} & \text{if } p^* = 1 \\ \frac{1}{1 - p_\theta(x)} & \text{if } p^* = 0 \end{cases} \\ &\therefore \frac{\partial \mathcal{L}_{\text{CE}}(p^*, p_\theta(x))}{\partial z_\theta(x)} = \begin{cases} p_\theta(x) - 1 & \text{if } p^* = 1 \\ p_\theta(x) & \text{if } p^* = 0 \end{cases} \\ &= p_\theta(x) - p^* \end{aligned}$$

Now we define gradient norm $g(x) \in [0, 1]$ as follow, which means the norm of gradient w.r.t $z_\theta(x)$:

$$g(x) = |p_\theta(x) - p^*| = \begin{cases} p^* - p_\theta(x) & \text{if } p^* = 1 \\ p_\theta(x) & \text{if } p^* = 0 \end{cases}$$

The value of $g(x)$ represents **attribute (e.g. easy or hard) of an example** and implies the example's impact on the global gradient.

很多现实分类任务(e.g. face detection.)存在和遇到的问题：**disharmony of gradient norm distribution**

- Easy examples:
 - the number of very easy examples(i.e. samples with very small $g(x)$) is extremely large, which have a great impact on the global gradient.
- Hard examples:

- a converged model $p_\theta(x)$ still can't handle some very hard examples(samples with large $g(x)$), whose number is even larger than the examples with medium difficulty(samples with medium $g(x)$).
- These very hard examples(samples with large $g(x)$) can be regarded as **outliers** since their gradient directions($p_\theta(x) - p^*$) tends to vary largely from the gradient directions of the large amount of other examples. That is, if the converged model is forced to learn to classify these outliers better, the classification of the large number of other examples tends to be less accurate.

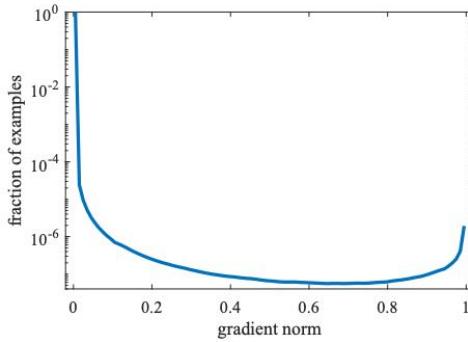


图 8: The distribution of the gradient norm $g(x)$ from a converged one-stage detection model. Note that the y-axis uses log scale since **easy negative samples have a dominant number**, and the number of examples with different gradient norm can differ by orders of magnitude. Cite from [Gradient Harmonized Single-stage Detector](#)[?]

Gradient Density

Definition 3.1 (gradient density function). 训练样本集上关于某一样本 (x, y) 的 gradient norm g 的梯度密度函数 (gradient density function) 可以表示为如下形式 :

$$\text{GD}(g) = \frac{1}{l_\epsilon(g)} \sum_{i=1}^N \delta_\epsilon(g(x^{(i)}), g)$$

其中, $g(x^{(i)})$ 为训练集中第 i 个样本的 gradient norm, 并且 :

$$\delta_\epsilon(x, y) = \begin{cases} 1 & \text{if } y - \frac{\epsilon}{2} \leq x < y + \frac{\epsilon}{2} \\ 0 & \text{otherwise.} \end{cases}$$

$$l_\epsilon(g) = \min(g + \frac{\epsilon}{2}, 1) - \max(g - \frac{\epsilon}{2}, 0)$$

对于 g 的梯度密度表示 :

- $\sum_{i=1}^N \delta_\epsilon(g(x^{(i)}), g)$: 训练集中 gradient norm 落在以 g 为中心, 以 ϵ 为最大区间直径的区间中的样本个数
- $\text{GD}(g) = \frac{1}{l_\epsilon(g)} \sum_{i=1}^N \delta_\epsilon(g(x^{(i)}), g)$: 将上述区间内样本个数 $\sum_{i=1}^N \delta_\epsilon(g(x^{(i)}), g)$ 除以区间长度 $l_\epsilon(g)$, 得到训练集中 gradient norm 在以 g 为中心, 以 ϵ 为最大区间直径的区间的 uniform probability distribution.

Definition 3.2 (gradient density harmonizing parameter).

$$\beta^{(i)} = \frac{N}{\text{GD}(g(x^{(i)}))}$$

其中 N 为训练集中训练样本的个数。为了更好地理解 gradient density harmonizing parameter $\beta^{(i)}$ ，可以将其重写为：

$$\beta^{(i)} = \frac{1}{\frac{\text{GD}(g(x^{(i)}))}{N}}$$

其中的分母 $\frac{\text{GD}(g(x^{(i)}))}{N}$ 表示所有训练样本中，gradient norm 大小在第 i 个样本的 gradient norm $g(x^{(i)})$ 附近的训练样本比例，因此可以发现，随着梯度密度 $\text{GD}(g(x^{(i)}))$ 的增加， $\frac{\text{GD}(g(x^{(i)}))}{N}$ 增加，相应的损失函数的权重 $\beta^{(i)}$ 就对应地减小。

因此 GHM 在分类和回归问题的目标函数为：

- GHM-C Loss:

$$\begin{aligned}\mathcal{L}_{\text{GHM-C}} &= \frac{1}{N} \sum_{i=1}^N \beta^{(i)} \mathcal{L}_{\text{CE}}(y^{(i)}, h_\theta(x^{(i)})) \\ &= \frac{1}{N} \sum_{i=1}^N \frac{1}{\text{GD}(g(x^{(i)}))} \mathcal{L}_{\text{CE}}(y^{(i)}, h_\theta(x^{(i)})) \\ &= \sum_{i=1}^N \frac{1}{\text{GD}(g(x^{(i)}))} \mathcal{L}_{\text{CE}}(y^{(i)}, h_\theta(x^{(i)}))\end{aligned}$$

因此：

- easy samples: largely down-weighted, address imbalance problem
- outliers(very hard samples): slightly down-weighted, address outliers problem
- medium samples: up-weighted, more focus!

3.3.7 Softmax + Cross entropy loss

Cross entropy for multi-class classification task is as follow:

$$\begin{aligned}
 \min_{\theta} J(\theta) &= \frac{1}{N} \sum_{i=1}^N \mathcal{L}(x^{(i)}, y^{(i)}; \theta) \\
 \Rightarrow \min_{\theta} J(\theta) &= \frac{1}{N} \sum_{i=1}^N \text{CE}(p^{(i)}, h_{\theta}(x^{(i)})) \quad (y^{(i)} \sim p^{(i)} \in \mathbb{P}^{[K]}) \\
 \Rightarrow \min_{\theta} J(\theta) &= \frac{1}{N} \sum_{i=1}^N -\langle p^{(i)}, \log h_{\theta}(x^{(i)}) \rangle \quad (h_{\theta}(x^{(i)}) \in \mathbb{P}^{[K]}) \\
 \Rightarrow \min_{\theta} J(\theta) &= -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log [h_{\theta}(x^{(i)})]_k \\
 \Rightarrow \min_{\theta} J(\theta) &= -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log \frac{e^{[z_{\theta}(x^{(i)})]_k}}{\sum_{j=1}^K e^{[z_{\theta}(x^{(i)})]_j}} \quad (\text{i.e. softmax function: } [h_{\theta}(x^{(i)})]_k = \frac{e^{[z_{\theta}(x^{(i)})]_k}}{\sum_{j=1}^K e^{[z_{\theta}(x^{(i)})]_j}} \in \mathbb{P})
 \end{aligned}$$

Where $[z_{\theta}(x^{(i)})]_k \in \mathbb{R}$ represents the **logit output(score)** of k -th class.

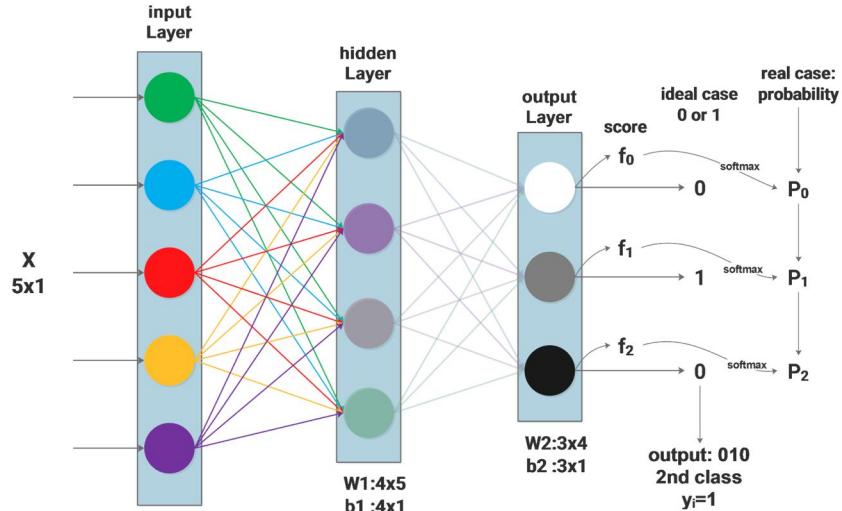


图 9: Softmax loss function. Cite from Wangxin's Blog: NN Softmax loss function

If the label probability distribution is **0-1 probability distribution**, we can rewrite softmax loss as follow:

$$\begin{aligned}
 \min_{\theta} J(\theta) &= -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log \frac{e^{[z_{\theta}(x^{(i)})]_k}}{\sum_{j=1}^K e^{[z_{\theta}(x^{(i)})]_j}} \\
 \Rightarrow \min_{\theta} J(\theta) &= -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K I(k = y^{(i)}) \log \frac{e^{[z_{\theta}(x^{(i)})]_k}}{\sum_{j=1}^K e^{[z_{\theta}(x^{(i)})]_j}} \quad (\text{i.e. 0-1 label probability distribution})
 \end{aligned}$$

Where the definition of indicator function is as follow:

$$I(k = y^{(i)}) = \begin{cases} 1 & \text{if } k = y^{(i)} \in \{1, \dots, K\}, \\ 0 & \text{otherwise.} \end{cases}$$

And we can rewrite softmax loss as follow:

$$\begin{aligned}
 \min_{\theta} J(\theta) &= -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K I(k = y^{(i)}) \log \frac{e^{[z_{\theta}(x^{(i)})]_k}}{\sum_{j=1}^K e^{[z_{\theta}(x^{(i)})]_j}} \\
 \Rightarrow \min_{\theta} J(\theta) &= -\frac{1}{N} \sum_{i=1}^N \log \frac{e^{[z_{\theta}(x^{(i)})]_{y^{(i)}}}}{\sum_{j=1}^K e^{[z_{\theta}(x^{(i)})]_j}} \\
 \Rightarrow \min_{\theta} J(\theta) &= -\frac{1}{N} \sum_{i=1}^N \log \frac{e^{W_{y^{(i)}}^T \cdot f_{\phi}(x^{(i)})}}{\sum_{j=1}^K e^{W_j^T \cdot f_{\phi}(x^{(i)})}} \quad (\text{seperate logit output layer and previous hidden layers})
 \end{aligned}$$

Where: $\theta = \{\phi, W\}$

- ϕ : parameters of all hidden layers.
- $W \in \mathbb{R}^{d \times K}$: parameters of last layer, mapping from **latent feature space** to **logit distribution space**.
- $f_{\phi}(x^{(i)}) \in \mathbb{R}^d$: latent feature vector of final hidden layer.
- $W_j \in \mathbb{R}^d$ means the j -th column of parameter matrix W , also:

$$[z_{\theta}(x^{(i)})]_j = W_j^T \cdot f_{\phi}(x^{(i)}) = \langle W_j^T, f_{\phi}(x^{(i)}) \rangle$$

Note: we omit the bias b_j in $[z_{\theta}(x^{(i)})]_j$ here to simplify analysis. Because $[z_{\theta}(x^{(i)})]_j$ is the inner product between W_j^T and $f_{\phi}(x^{(i)})$, therefore it can be also formulated as follow:

$$\begin{aligned}
 \min_{\phi, W} J(\phi, W) &= -\frac{1}{N} \sum_{i=1}^N \log \frac{e^{W_{y^{(i)}}^T \cdot f_{\phi}(x^{(i)})}}{\sum_{j=1}^K e^{W_j^T \cdot f_{\phi}(x^{(i)})}} \\
 \Rightarrow \min_{\phi, W} J(\phi, W) &= -\frac{1}{N} \sum_{i=1}^N \log \frac{e^{\|W_{y^{(i)}}^T\| \|f_{\phi}(x^{(i)})\| \cos(\theta_{y^{(i)}, i})}}{\sum_{j=1}^K e^{\|W_j^T\| \|f_{\phi}(x^{(i)})\| \cos(\theta_{j, i})}} \quad (\because W_j^T \cdot f_{\phi}(x^{(i)}) = \|W_j^T\| \|f_{\phi}(x^{(i)})\| \cos(\theta_{j, i}))
 \end{aligned}$$

Where $\theta_{j,i} \in [0, \pi]$ ($j \in \{1, \dots, K\}$) is the angle between the parameter vector W_j^T and feature vector $f_{\phi}(x^{(i)})$.

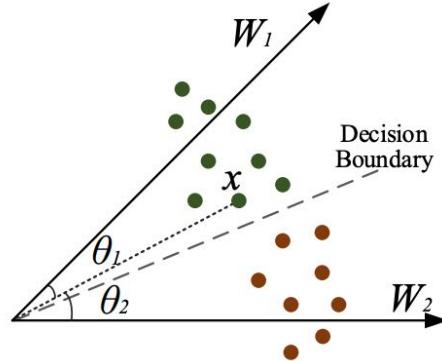
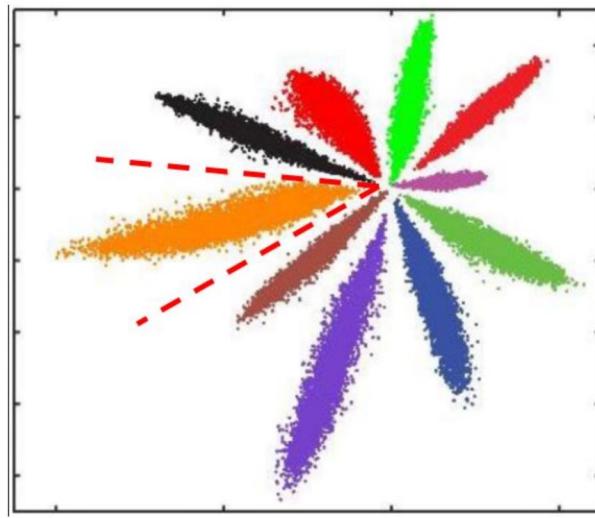


图 10: $\theta_{j,i} \in [0, \pi]$ ($j \in \{1, \dots, K\}$) is the angle between the parameter vector W_j^T and feature vector $f_{\phi}(x^{(i)})$.

How to modify Softmax loss? 以 MNIST 作为 dataset，CNN 作为 model inference, Softmax Loss 作为 loss function 为例：MNIST 上 10 分类的 2 维特征映射可视化如下：

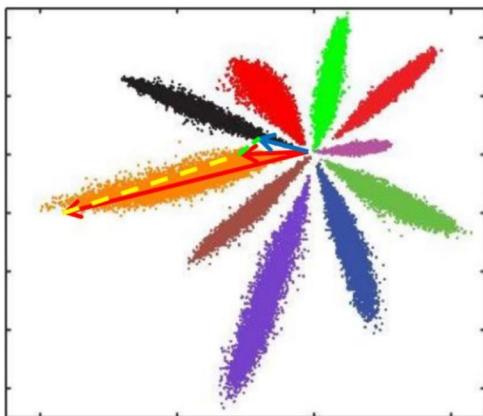


不同类别之间的抽取特征向量 $f_\phi(\mathbf{x})$ 明显分隔开了，但这种情况并不满足很多应用场景（e.g. 人脸识别）中特征向量对比差异的需求。因为特征向量相似度的计算常用：

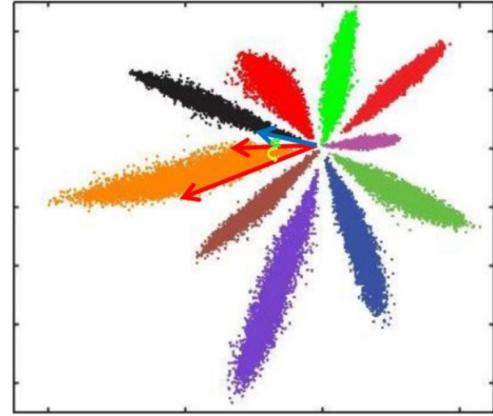
- 欧式距离 (L2 distance)
- 余弦距离 (cosine distance)

所以在此分别讨论两种情况下遇到的问题。

- 欧式距离 (L2 distance)：L2 距离越小，向量相似度越高。但是可能同类的特征向量距离（黄色）比不同类的特征向量距离（绿色）更大。
- 余弦距离 (cosine distance)：夹角越小，cos 距离越大，向量相似度越高。但是可能同类的特征向量夹角（黄色）比不同类的特征向量夹角（绿色）更大



(a) l2 distance



(b) cos distance

因此总结来讲：

- Softmax Loss 训练的深度特征，会把整个超空间或者超球，按照分类个数进行划分，保证类别是可分的，这一点对一般多分类任务如 MNIST 和 ImageNet 非常合适，因为测试类别必定在训练类别中。
- Softmax Loss 存在的问题是：**不要求类内紧凑和类间分离**，这一点非常不适合复杂的多分类任务 (e.g. 人脸识别)。因为训练集的 10000 人数 (10000 类)，相对测试集整个世界 70 亿人类 (70 亿类) 来说，非常微不足道，而我们不可能拿到所有人的训练样本，更过分的是，一般我们还要求训练集和测试集不重叠。
- 因此需要改造 Softmax Loss，指导思想是：**除保证可分性外，还要做到特征向量 $f_\phi(x)$ 类内尽可能紧凑，类间尽可能分离。**
- 对 Softmax loss 的改造有如下几类：
 1. Margin-Loss: insert a geodesic distance margin between the feature sample $f_\phi(\mathbf{x})$ and the corresponding centre \mathbf{W}_y .
 - Large-Margin Softmax Loss[?]
 - Angular-Softmax Loss[?]
 - ArcFace Loss(Additive Angular-Margin Loss)[?]
 2. Intra-Loss: decrease the geodesic distance between the feature sample $f_\phi(\mathbf{x})$ and the corresponding centre \mathbf{W}_y .
 3. Inter-Loss: increase the geodesic distance between different centres(i.e. $\mathbf{W}_{j_1}, \mathbf{W}_{j_2}, \forall j_1, j_2 \in \{1, \dots, K\}, j_1 \neq j_2$).
 4. Triplet-Loss: insert a geodesic distance margin between triplet samples.

因此改造 Softmax Loss 的模版形式如下：

$$\begin{aligned} \mathcal{L}(x^{(i)}, y^{(i)}, \phi, W) = & -\log \underbrace{\left(\frac{e^{\|W_{y^{(i)}}^\top\| \|f_\phi(x^{(i)})\| \cos(m_1 \cdot \theta_{y^{(i)}, i} + m_2)}}{e^{\|W_{y^{(i)}}^\top\| \|f_\phi(x^{(i)})\| \cos(m_1 \cdot \theta_{y^{(i)}, i} + m_2)} + \sum_{j=1, j \neq y^{(i)}}^K e^{\|W_j^\top\| \|f_\phi(x^{(i)})\| \cos(\theta_{j, i})}} \right)}_{\text{Margin-Loss}} \\ & + \underbrace{\|f_\phi(x^{(i)}) - W_{y^{(i)}}\|}_{\text{Intra-Loss}} \\ & - \underbrace{\frac{1}{K-1} \sum_{j=1, j \neq y^{(i)}}^K \|W_{y^{(i)}}^\top - W_j\|}_{\text{Inter-Loss}} \end{aligned}$$

3.3.8 Large-Margin Softmax + Cross entropy loss (L-Softmax)

论文出处：Large-Margin Softmax Loss for Convolutional Neural Networks[?]. 该目标函数强制让样本特征向量 $f_\phi(x) \in \mathbb{R}^d$ 和对应类别的参数向量 W_y 之间的夹角增加到原来 Softmax loss 的 m 倍。

Intuition softmax loss 如下：

$$\min_{\phi, W} J(\phi, W) = -\frac{1}{N} \sum_{i=1}^N \log \frac{e^{\|W_{y(i)}^\top\| \|f_\phi(x^{(i)})\| \cos(\theta_{y(i), i})}}{\sum_{j=1}^K e^{\|W_j^\top\| \|f_\phi(x^{(i)})\| \cos(\theta_{j, i})}}$$

Where $\theta_{j,i} \in [0, \pi]$ ($j \in \{1, \dots, K\}$) is the angle between the parameter vector W_j^\top and feature vector $f_\phi(x^{(i)})$.

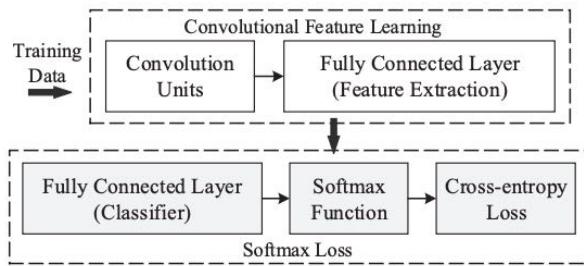


图 11: Standard CNNs can be viewed as convolutional feature learning machines that are supervised by the softmax loss.

Consider the binary classification:

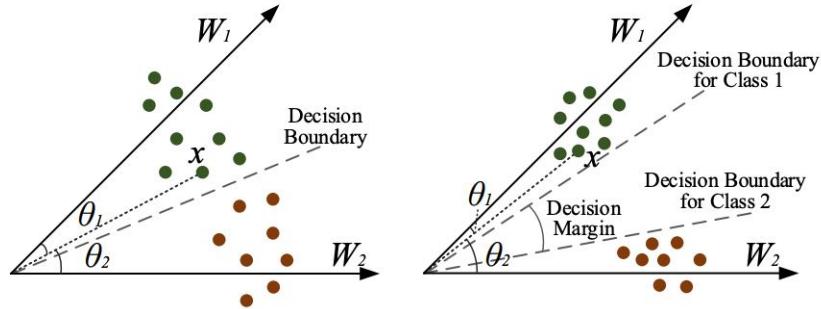


图 12: Original Softmax Loss and L-Softmax Loss

- Suppose we have a sample $f_\phi(x)$ from 1-th class.
- The original softmax is to force:

$$W_1^T x > W_2^T x$$

namely:

$$\|W_1\| \|x\| \cos(\theta_1) > \|W_2\| \|x\| \cos(\theta_2)$$

in order to classify x correctly.

- However, we want to make the classification more rigorous in order to produce a decision margin.
- So we instead require:

$$\|\mathbf{W}_1^T\| \|\mathbf{x}\| \cos(m\theta_1) > \|\mathbf{W}_2^T\| \|\mathbf{x}\| \cos(\theta_2) \quad (0 \leq \theta_1 \leq \frac{\pi}{m})$$

where m is a **positive integer**.

Theorem 1 (Inequality of Large-Margin Softmax Loss). Because the following inequality holds:

$$\begin{aligned} \|\mathbf{W}_1^T\| \|\mathbf{x}\| \cos(\theta_1) &> \|\mathbf{W}_2^T\| \|\mathbf{x}\| \cos(m\theta_1) \\ &> \|\mathbf{W}_2^T\| \|\mathbf{x}\| \cos(\theta_2) \end{aligned}$$

Therefore, $\|\mathbf{W}_1^T\| \|\mathbf{x}\| \cos(\theta_1) > \|\mathbf{W}_2^T\| \|\mathbf{x}\| \cos(\theta_2)$ has to hold. So the new classification criteria is a stronger requirement to correctly classify \mathbf{x} , producing a more rigorous decision boundary for 1-th class.

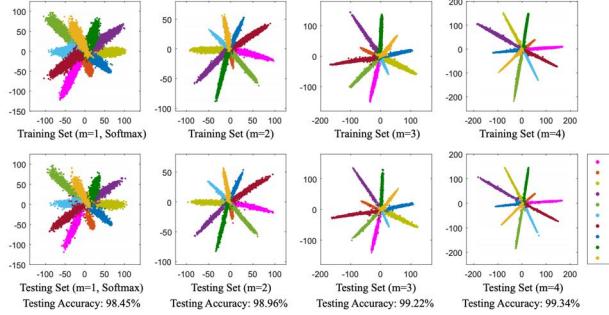


图 13: CNN-learned features visualization (Softmax Loss ($m = 1$) vs. L-Softmax loss ($m = 2, 3, 4$)) in MNIST dataset. Specifically, we set the feature (input of the L-Softmax loss) dimension as 2, and then plot them by class. We omit the constant term in the fully connected layer, since it just complicates our analysis and nearly does not affect the performance.

Definition

Definition 3.3 (L-Softmax loss). the L-Softmax loss is defined as:

$$\mathcal{L}(x^{(i)}, y^{(i)}; W, \phi) = -\log \left(\underbrace{\frac{e^{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\| \psi(\theta_{y^{(i)}, i})}}{\underbrace{e^{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\| \psi(\theta_{y^{(i)}, i})} + \sum_{j=1, j \neq y^{(i)}}^K e^{\|W_j^T\| \|f_\phi(x^{(i)})\| \cos(\theta_{j, i})}}}_{\text{positive class}} \right)$$

in which we generally require:

$$\psi(\theta) = \begin{cases} \cos(m\theta), & 0 \leq \theta \leq \frac{\pi}{m} \\ \mathcal{D}(\theta), & \frac{\pi}{m} \leq \theta \leq \pi \end{cases}$$

where:

- m is a positive integer that is closely related to the classification margin.
- with larger m , the classification margin becomes larger and the learning objective also becomes harder.
- $\mathcal{D}(\theta)$ (e.g. $\cos(\theta)$) is required to be a monotonically decreasing function and $\mathcal{D}(\frac{\pi}{m})$ should equal $\cos(\frac{\pi}{m})$.
- To simplify the forward and backward propagation, the author construct a specific $\psi(\theta)$ in the paper:

$$\psi(\theta) = (-1)^k \cos(m\theta) - 2k, \quad \theta \in [\frac{k\pi}{m}, \frac{(k+1)\pi}{m}]$$

where $k \in [0, m-1]$ and k is an integer.

但是 L-Softmax loss 存在缺陷：

1. 没有对参数向量 $W_j, \forall j \in \{1, \dots, K\}$ 进行归一化 (normalization)，导致的结果是：
 - 由于 $\|W_{y(i)}\|$ 很大，所以即使 $W_{y(i)}$ 和 $f_\phi(x^{(i)})$ 之间的夹角 $\theta_{y(i),i}$ 不用很小，也可以使得向量内积 $W_{y(i)} \cdot f_\phi(x^{(i)})$ 的值很大，从而影响类别的判别
 - 而我们真正希望优化参数 $\{\phi, W\}$ 得到的结果是向量 $W_{y(i)}$ 和 $f_\phi(x^{(i)})$ 之间的夹角 $\theta_{y(i),i}$ 非常小！
2. 没有对隐特征向量 $f_\phi(x)$ 进行归一化 (normalization)，导致的结果同上，没有完全专注于训练向量的方向，而有可能是向量长度。因此根据论文L2-constrained Softmax Loss for Discriminative Face Verification[?]，对 $f_\phi(x)$ 进行归一化操作。

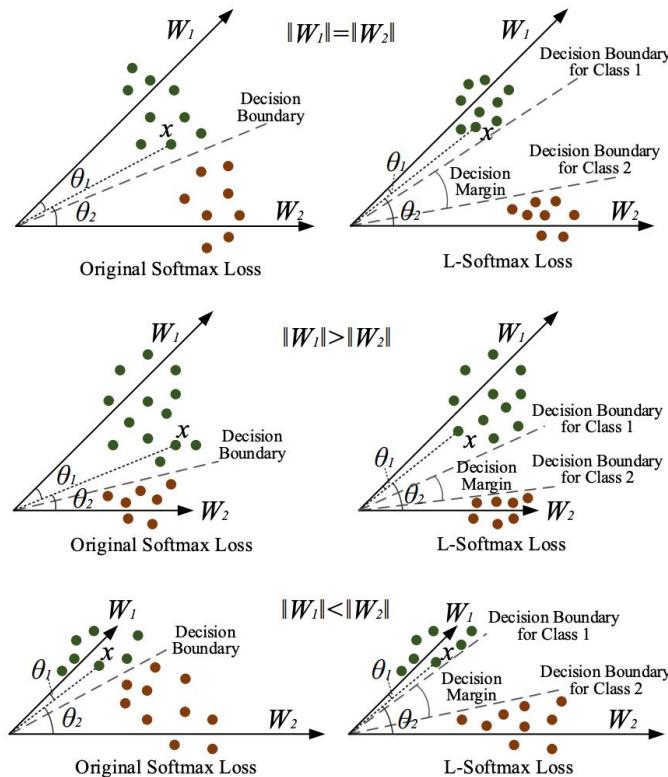


图 14: Examples of Geometric Interpretation

Optimization (未完成)

- $\nabla_{W_{y^{(i)}}} \mathcal{L}(x^{(i)}, y^{(i)}; W, \phi)$
- $\nabla_{W_j} \mathcal{L}(x^{(i)}, y^{(i)}; W, \phi) \quad (j \neq y^{(i)})$

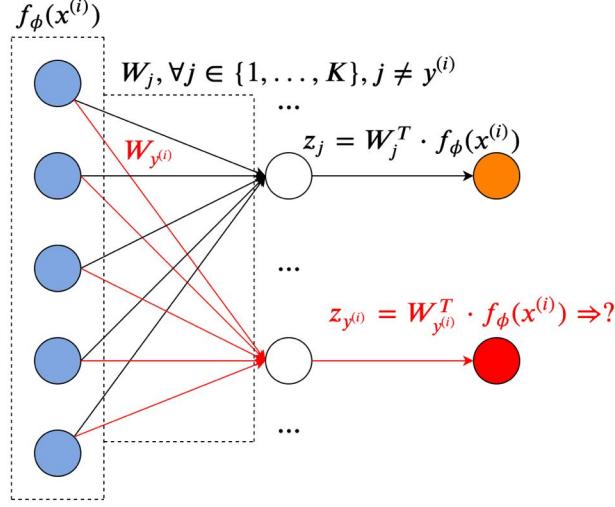


图 15: Optimization for Large-margin softmax + cross-entropy loss.

Proof. 正样本上的逻辑输出 (logit output) 正向推导表示如下：

$$\begin{aligned}
 z_{y^{(i)}} &= W_{y^{(i)}}^T f_\phi(x^{(i)}) \\
 &= \|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\| \cos(\theta_{y^{(i)}, i}) \\
 &\Rightarrow \|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\| \psi(\theta_{y^{(i)}, i}) \quad (\text{修正原始 softmax 函数中正样本对应的夹角}) \\
 &= \|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\| \left((-1)^k \cos(m\theta_{y^{(i)}, i}) - 2k \right) \quad (\because \psi(\theta) = (-1)^k \cos(m\theta) - 2k, \quad \theta \in [\frac{k\pi}{m}, \frac{(k+1)\pi}{m}]) \\
 &= (-1)^k \cdot \|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\| \cos(m\theta_{y^{(i)}, i}) - 2k \cdot \|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\| \\
 \because \cos(m\theta) &= C_m^0 \cos^m(\theta) - C_m^2 \cos^{m-2}(\theta) \sin^2(\theta) + C_m^4 \cos^{m-4}(\theta) \sin^4(\theta) + \dots + \underbrace{(-1)^n C_m^{2n} \cos^{m-2n}(\theta) \sin^{2n}(\theta)}_{\text{标准项}} + \dots \\
 &= C_m^0 \cos^m(\theta) - C_m^2 \cos^{m-2}(\theta)(1 - \cos^2(\theta)) + C_m^4 \cos^{m-4}(\theta)(1 - \cos^2(\theta))^2 + \dots \\
 &\quad + \underbrace{(-1)^n C_m^{2n} \cos^{m-2n}(\theta)(1 - \cos^2(\theta))^n}_{\text{标准项}} + \dots \quad (2n \leq m \text{ 并且 } n \text{ 为整数.}) \\
 \therefore \cos(\theta_{y^{(i)}, i}) &= \frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \quad (\theta_{y^{(i)}, i} \text{ 为正样本嵌入向量 } f_\phi(x^{(i)}) \text{ 和对应参数 } W_{y^{(i)}} \text{ 之间的夹角}) \\
 \therefore z_{y^{(i)}} &= (-1)^k \cdot \|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\| \left(C_m^0 \cos^m(\theta_{y^{(i)}, i}) - C_m^2 \cos^{m-2}(\theta_{y^{(i)}, i}) \sin^2(\theta_{y^{(i)}, i}) \right. \\
 &\quad \left. + C_m^4 \cos^{m-4}(\theta_{y^{(i)}, i}) \sin^4(\theta_{y^{(i)}, i}) + \dots + (-1)^n C_m^{2n} \cos^{m-2n}(\theta_{y^{(i)}, i}) \sin^{2n}(\theta_{y^{(i)}, i}) + \dots \right) \\
 &\quad - 2k \cdot \|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|
 \end{aligned}$$

$$\begin{aligned}
z_{y^{(i)}} &= (-1)^k \cdot \|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\| \left(C_m^0 \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^m \right. \\
&\quad - C_m^2 \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^{m-2} \left(1 - \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^2 \right) \\
&\quad + C_m^4 \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^{m-4} \left(1 - \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^2 + \dots \\
&\quad \left. + (-1)^n C_m^{2n} \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^{m-2n} \left(1 - \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^n + \dots \right) - 2k \cdot \|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|
\end{aligned}$$

Consider the gradient of one specific term and note as:

$$\begin{aligned}
T_n(W_{y^{(i)}}, f_\phi(x^{(i)})) &= (-1)^n C_m^{2n} \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^{m-2n} \left(1 - \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^n \\
\therefore \frac{\partial}{\partial f_\phi(x^{(i)})} \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^{m-2n} &= \frac{(m-2n)(W_{y^{(i)}}^T f_\phi(x^{(i)}))^{m-2n-1} W_{y^{(i)}}}{(\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|)^{m-2n-1}} \\
\therefore \frac{\partial}{\partial f_\phi(x^{(i)})} \left(1 - \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^n &= (-2n) \cdot \left(1 - \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^{n-1} \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right) W_{y^{(i)}} \\
\therefore \frac{\partial T_n(W_{y^{(i)}}, f_\phi(x^{(i)}))}{\partial f_\phi(x^{(i)})} &= (-1)^n C_m^{2n} \left(\frac{(m-2n)(W_{y^{(i)}}^T f_\phi(x^{(i)}))^{m-2n-1} W_{y^{(i)}}}{(\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|)^{m-2n-1}} \left(1 - \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^n \right. \\
&\quad \left. + \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^{m-2n} (-2n) \cdot \left(1 - \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^{n-1} \left(\frac{W_{y^{(i)}}^T f_\phi(x^{(i)})}{\|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\|} \right) W_{y^{(i)}} \right)
\end{aligned}$$

Reference

- [github: Large-Margin Softmax Loss](#)
- [Billy's Blog: <Tensorflow> How to implement the larget margin softmax loss in tensorflow](#)
- [知乎：人脸识别的 LOSS（上）](#)
- [知乎：人脸识别的 LOSS（下）](#)

3.3.9 Additive Margin Softmax + Cross entropy loss (AM-Softmax)

论文出处：[Additive Margin Softmax for Face Verification\[?\]](#)

3.3.10 Angular-Softmax + Cross entropy loss (A-Softmax)

论文出处：[SphereFace: Deep Hypersphere Embedding for Face Recognition](#)[?]. 该目标函数相对于L-Softmax loss[?]归一化了权重 W ，让训练更加集中在优化深度特征映射 $f_\phi(x)$ 和特征向量 W_j 的角度上，降低样本数量不均衡问题。

Definition

$$\begin{aligned}
 \mathcal{L}(x^{(i)}, y^{(i)}; W, \phi) &= \text{CE}(p^{(i)}, h_{\phi, W}(x^{(i)})) \quad (y^{(i)} \sim p^{(i)} \in \mathbb{P}^{|K|}) \\
 &= -\left\langle p^{(i)}, \log(h_{\phi, W}(x^{(i)})) \right\rangle \quad (h_{\phi, W}(x^{(i)}) \in \mathbb{P}^{|K|}) \\
 &= -\sum_{k=1}^K p_k^{(i)} \log(h_{\phi, W_k}(x^{(i)})) \quad (W_k \text{ is the } k\text{-th column of } W, h_{\phi, W_k}(x^{(i)}) \in \mathbb{P}) \\
 &= -\sum_{k=1}^K p_k^{(i)} \log\left(\frac{e^{z_{\phi, W_k}(x^{(i)})}}{\sum_{j=1}^K e^{z_{\phi, W_j}(x^{(i)})}}\right) \quad (\text{softmax function, } z_{\phi, W_j}(x) \in \mathbb{R} \text{ outputs logit score}) \\
 &= -\sum_{k=1}^K p_k^{(i)} \log\left(\frac{e^{W_k^T \cdot f_\phi(x^{(i)}) + b_k}}{\sum_{j=1}^K e^{W_j^T \cdot f_\phi(x^{(i)}) + b_j}}\right) \quad (\text{unpack classifier fully connected layer}) \\
 &= -\sum_{k=1}^K p_k^{(i)} \log\left(\frac{e^{\|W_k\| \|f_\phi(x^{(i)})\| \cos(\theta_{k,i}) + b_k}}{\sum_{j=1}^K e^{\|W_j\| \|f_\phi(x^{(i)})\| \cos(\theta_{j,i}) + b_j}}\right) \\
 &= -\sum_{k=1}^K p_k^{(i)} \log\left(\frac{e^{\|f_\phi(x^{(i)})\| \cos(\theta_{k,i}) + b_k}}{\sum_{j=1}^K e^{\|f_\phi(x^{(i)})\| \cos(\theta_{j,i}) + b_j}}\right) \quad (\text{constraint: normalize } \|W_j\| = 1, \forall j \in \{1, \dots, K\}) \\
 &= -\sum_{k=1}^K p_k^{(i)} \log\left(\frac{e^{\|f_\phi(x^{(i)})\| \cos(\theta_{k,i})}}{\sum_{j=1}^K e^{\|f_\phi(x^{(i)})\| \cos(\theta_{j,i})}}\right) \quad (\text{zero the bias}) \\
 &= -\sum_{k=1}^K \mathbf{I}(k = y^{(i)}) \log\left(\frac{e^{\|f_\phi(x^{(i)})\| \cos(\theta_{k,i})}}{\sum_{j=1}^K e^{\|f_\phi(x^{(i)})\| \cos(\theta_{j,i})}}\right) \quad (\text{using 0-1 label probability distribution}) \\
 &= -\log\left(\frac{e^{\|f_\phi(x^{(i)})\| \cos(\theta_{y^{(i)}, i})}}{\sum_{j=1}^K e^{\|f_\phi(x^{(i)})\| \cos(\theta_{j,i})}}\right)
 \end{aligned}$$

在此 normalize $W_j, \forall j$ 并且 zero $b_j \forall j$ 基础上，再利用 L-Softmax Loss 的方法，可得：

$$\mathcal{L}_{\text{ang}}(x^{(i)}, y^{(i)}; W, \phi) = -\log\left(\frac{e^{\|f_\phi(x^{(i)})\| \cos(m\theta_{y^{(i)}, i})}}{e^{\|f_\phi(x^{(i)})\| \cos(m\theta_{y^{(i)}, i})} + \sum_{j=1, j \neq y^{(i)}}^K e^{\|f_\phi(x^{(i)})\| \cos(\theta_{j,i})}}\right)$$

其中超参数 m 为大于 0 的正整数，并且向量 $W_{y^{(i)}}$ 和 $f_\phi(x^{(i)})$ 之间的夹角 $\theta_{y^{(i)}, i}$ 必须满足：

$$\theta_{y^{(i)}, i} \in \left[0, \frac{\pi}{m}\right]$$

将上述公式中的 $\cos(m\theta_{y^{(i)}, i})$ 泛化为函数 $\psi(\theta_{y^{(i)}, i})$ ，并且该函数满足：

- $\psi(\theta_{y^{(i)}, i})$ 随着因变量 $\theta_{y^{(i)}, i}$ 单调递减
- 在 $[0, \frac{\pi}{m}]$ 区间， $\psi(\theta_{y^{(i)}, i}) = \cos(m\theta_{y^{(i)}, i})$

因此泛化版的 A-Softmax loss 可以表示如下：

$$\mathcal{L}_{\text{ang}}(x^{(i)}, y^{(i)}; W, \phi) = -\log\left(\frac{e^{\|f_\phi(x^{(i)})\| \psi(\theta_{y^{(i)}, i})}}{e^{\|f_\phi(x^{(i)})\| \psi(\theta_{y^{(i)}, i})} + \sum_{j=1, j \neq y^{(i)}}^K e^{\|f_\phi(x^{(i)})\| \cos(\theta_{j,i})}}\right)$$

in which we define

$$\psi(\theta_{y^{(i)}, i}) = (-1)^k \cos(m\theta_{y^{(i)}, i}) - 2k, \quad \theta_{y^{(i)}, i} \in \left[\frac{k\pi}{m}, \frac{(k+1)\pi}{m} \right]$$

for all $k \in [0, m-1]$, ($m \geq 1$). 最终的结果如图所示：

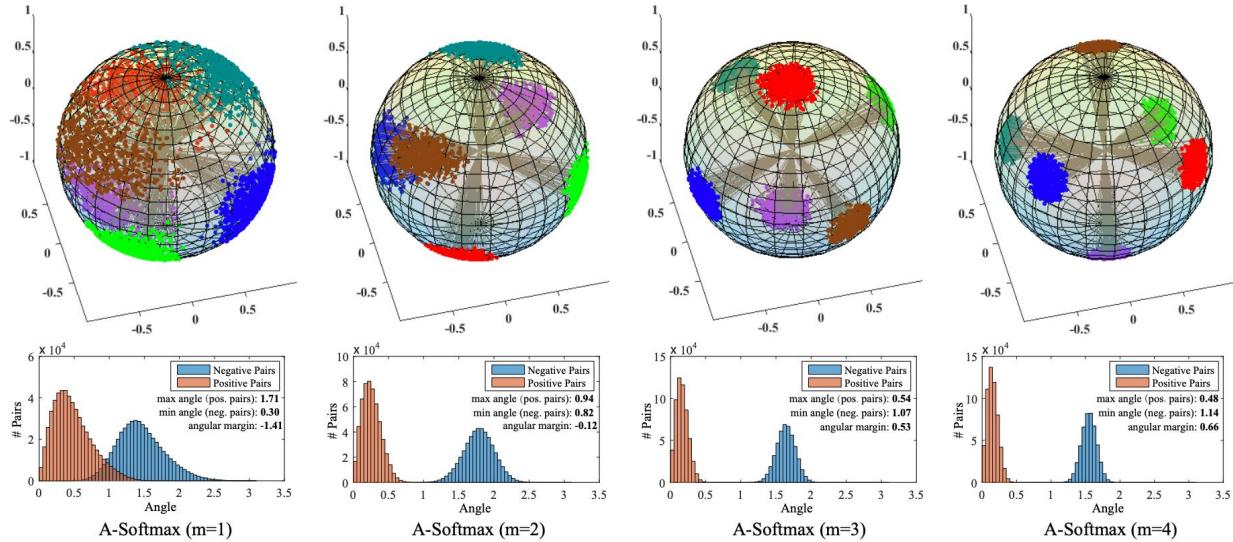


图 16: Visualization of features learned with different m . The first row shows the 3D features projected on the unit sphere. The projected points are the intersection points of the feature vectors and the unit sphere. The second row shows the angle distribution of both positive pairs and negative pairs (we choose class 1 and class 2 from the subset to construct positive and negative pairs). Orange area indicates positive pairs while blue indicates negative pairs.

注意：A-Softmax loss[?] 和 L-Softmax loss[?] 都使用了乘性 margin 使不同类别的特征 $f_\phi(x) \in \mathbb{R}^d$ 更加分离，并且特征相似度的计算都采用了 cos 距离。但需要注意的是，这两个 loss 如果直接训练很难收敛，因此实际中都采用了退火优化策略 (annealing optimization strategy)，logit output 的表示如下：

$$z_{y^{(i)}} = \frac{\lambda \|W_{y^{(i)}}\| \|f_\phi(x)\| \cos(\theta_{y^{(i)}, i}) + \|W_{y^{(i)}}\| \|f_\phi(x)\| \psi(\theta_{y^{(i)}, i})}{\lambda + 1}$$

即逐步由相对容易训练的 Softmax Loss 退火到相对困难训练的**注意：A-Softmax loss[?] 或 L-Softmax loss[?]**。

Reference

- A Performance Comparison of Loss Functions for Deep Face Recognition[?]
- [github: Angular-Softmax loss](#)

3.3.11 ArcFace + Cross entropy loss

论文出处：[ArcFace: Additive Angular Margin Loss for Deep Face Recognition\[?\]](#)

Definition

$$\mathcal{L}_{\text{arc}}(x^{(i)}, y^{(i)}; W, \phi) = -\log \left(\frac{e^{\alpha \|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\| \cos(\theta_{y^{(i)}, i} + m)}}{e^{\alpha \|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\| \cos(\theta_{y^{(i)}, i} + m)} + \sum_{j=1, j \neq y^{(i)}}^K e^{\alpha \|W_j^T\| \|f_\phi(x^{(i)})\| \cos(\theta_{j, i})}} \right)$$

为了保障 W_j^T 和 $f_\phi(x)$ 的学习都是在方向上而不是在无用的长度上，因此在计算前先对 W_j^T 和 $f_\phi(x)$ 进行归一化操作：

$$\begin{aligned}\|W_{y^{(i)}}^T\| &= 1 \\ \|f_\phi(x^{(i)})\| &= 1\end{aligned}$$

而后为了保障训练速度和效果，添加了一个**re-scale factor**: $\alpha \in (0, +\infty)$

$$s(y^{(i)}, i) \in \mathbb{R} = \alpha \cdot \|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\| = \alpha$$

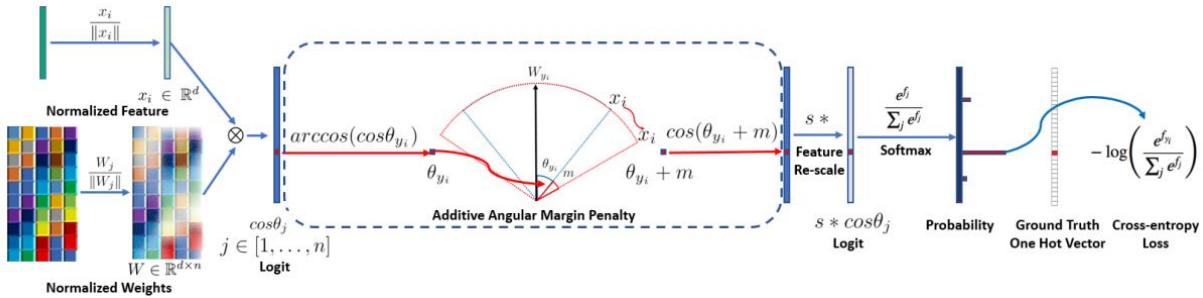


图 17: Training a DCNN for face recognition supervised by the ArcFace loss. Based on the feature $x^{(i)}$ and weight W normalization, we get the $\cos(\theta_{j,i})$ (logit) for each class as $W_j^T x^{(i)}$. We calculate the $\arccos(\theta_{y^{(i)},i})$ and get the angle between the feature $x^{(i)}$ and the ground truth weight $W_{y^{(i)}}$. In fact, W_j provides a kind of center for each class. Then, we add an angular margin penalty m on the target (ground truth) angle $\theta_{y^{(i)},i}$. After that, we calculate $\cos(\theta_{y^{(i)},i} + m)$ and multiply all logits by the feature scale s . The logits then go through the softmax function and contribute to the cross entropy loss.

因此真正的目标函数为：

$$\min_{\phi, W} J(\phi, W) = -\frac{1}{N} \sum_{i=1}^N \log \left(\frac{e^{s(y^{(i)}, i) \cos(\theta_{y^{(i)}, i} + m)}}{e^{s(y^{(i)}, i) \cos(\theta_{y^{(i)}, i} + m)} + \sum_{j=1, j \neq y^{(i)}}^K e^{s(j, i) \cos(\theta_{j, i})}} \right)$$

Idea

- Margin-Loss 的模版：综合对正样本夹角 $\theta_{y^{(i)}, i}$ 的加和乘： $\cos(\theta_{y^{(i)}, i} + m) \Rightarrow \cos(m_1 \cdot \theta_{y^{(i)}, i} + m_2)$
- 不同样本的 $s(y^{(i)}, i) = \alpha \cdot \|W_{y^{(i)}}^T\| \|f_\phi(x^{(i)})\| (\alpha > 0)$ 中对应的 α 尺度应该不同
- 融合 online hard sample mining(e.g. focal loss)

3.3.12 Triplet loss

论文出处：[FaceNet: A Unified Embedding for Face Recognition and Clustering](#) [?]。该类目标函数用于**细粒度的个体识别**应用场景。并且 Triplet loss 用于**embedding** $f_\phi(x) \in \mathbb{R}^D$ 的学习，而之后没有 softmax 层，也就没有分类参数 $W \in \mathbb{R}^{D \times K}$ 的学习。

Introduction Triplet loss 的引入需要从传统分类任务的损失函数 softmax + cross-entropy 说起，而传统 softmax + cross-entropy 分类训练结构 + 损失函数的框架存在如下问题：

1. 假设每一个目标 (identity) 的 embedding 维度为 D ，即： $f_\phi(x) \in \mathbb{R}^D$ ；在分类**类别数量固定**的情况下，总分类个数为 K ，那么仅 softmax 层的参数数量共计 $D \times K$ 个，如果类别数量非常庞大 (e.g. 全中国人的人脸识别，13 亿个类)，则算法无论从参数设置存储角度，还是参数的训练学习角度看都非常不现实。
2. 在一些识别任务中，**类别个数 K 为变量而非固定值**，譬如在人脸识别任务中，有可能这次识别目标规模小，需要在 $K = 100$ 个类别中进行识别比较；而下一次识别规模较大，需要在包含上次所有类别的 $K = 10000$ 个类别中进行比较，那么这一次有 $10000 - 100 = 9900$ 个类没有被训练过，不靠谱。
3. 很多时候特别在**细粒度识别任务**中，识别比较更多地是“**相对优势**”ranking 的思想而非“**非黑即白**”的 classification 的思想，譬如 NLP 领域的近似文本匹配任务，并不是一个非黑即白的二分类任务，更多地是模型 rank 出一个结果，看谁相对于其他更加匹配。

Methodology $(x^{(A)} \in \mathbb{R}^X, x^{(P)} \in \mathbb{R}^X, x^{(N)} \in \mathbb{R}^X)$ 的三元组为一个 sample，并且从样本空间到 embedding 空间的映射函数为 $f_\phi(\cdot) : \mathbb{R}^X \rightarrow \mathbb{R}^D$ 。Triplet loss 的目的如下：

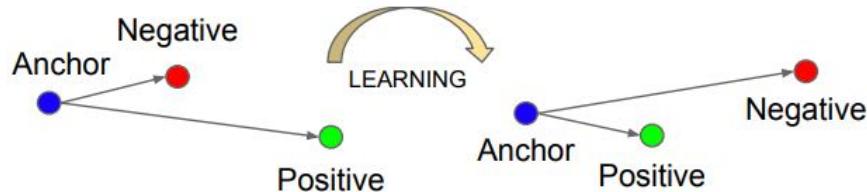


图 18: The Triplet Loss minimizes the distance between an anchor and a positive, both of which have the same identity, and maximizes the distance between the anchor and a negative of a different identity.

- 最小化 Anchor (baseline) 和 Positive 的**绝对**距离：

$$\min_{\phi} d(f_\phi(x^{(A)}), f_\phi(x^{(P)}))$$

- 最大化 Anchor (baseline) 和 Negative 的**绝对**距离：

$$\begin{aligned} & \max_{\phi} d(f_\phi(x^{(A)}), f_\phi(x^{(N)})) \\ \Rightarrow & \min_{\phi} -d(f_\phi(x^{(A)}), f_\phi(x^{(N)})) \end{aligned}$$

- (难理解点) 保障 Anchor (baseline) 和 Positive 之间的距离相对小于 Anchor (baseline) 和 Negative 之间的距离：

$$\begin{aligned}
 & d(f_\phi(x^{(A)}), f_\phi(x^{(P)})) < d(f_\phi(x^{(A)}), f_\phi(x^{(N)})) \\
 \Rightarrow & d(f_\phi(x^{(A)}), f_\phi(x^{(P)})) - d(f_\phi(x^{(A)}), f_\phi(x^{(N)})) < 0 \\
 \Rightarrow & d(f_\phi(x^{(A)}), f_\phi(x^{(P)})) - d(f_\phi(x^{(A)}), f_\phi(x^{(N)})) < 0 - \alpha \quad (\text{为 margin 超参数, 更加严格}) \\
 \Rightarrow & d(f_\phi(x^{(A)}), f_\phi(x^{(P)})) - d(f_\phi(x^{(A)}), f_\phi(x^{(N)})) + \alpha < 0
 \end{aligned}$$

α 的存在是为了尽可能限制 $d(f_\phi(x^{(A)}), f_\phi(x^{(P)})) > d(f_\phi(x^{(A)}), f_\phi(x^{(N)}))$ 。又因为 $d(\cdot, \cdot) \geq 0$ 恒成立，因此我们希望上述距离之差的值尽可能小（实际上最终越负数越好），即：

$$\min_{\phi} d(f_\phi(x^{(A)}), f_\phi(x^{(P)})) - d(f_\phi(x^{(A)}), f_\phi(x^{(N)})) + \alpha$$

但是从训练阶段优化的角度看，分两类情况：

- 当 $d(f_\phi(x^{(A)}), f_\phi(x^{(P)})) - d(f_\phi(x^{(A)}), f_\phi(x^{(N)})) + \alpha < 0$ 时：即 Anchor 到 Positive 之间的距离已经比 Anchor 到 Negative 之间的距离小 α 了，已经很好地满足了要求了，所以不必考虑这个三元组样本了，直接将这个样本对应的损失 mask 掉（损失函数值设置为 0）。
- 当 $d(f_\phi(x^{(A)}), f_\phi(x^{(P)})) - d(f_\phi(x^{(A)}), f_\phi(x^{(N)})) + \alpha \geq 0$ 时：可视为 Anchor 到 Positive 之间的距离还比 Anchor 到 Negative 大，即不满足条件，仍然需要优化学习。

综上所述，目标函数最终设置为：

$$\mathcal{L}(\phi; (x^{(A)}, x^{(P)}, x^{(N)})) = \max(d(f_\phi(x^{(A)}), f_\phi(x^{(P)})) - d(f_\phi(x^{(A)}), f_\phi(x^{(N)})) + \alpha, 0)$$

注意 上述距离 $d(\cdot, \cdot)$ (e.g. $d(f_\phi(x^{(A)}), f_\phi(x^{(P)})) = \|f_\phi(x^{(A)}) - f_\phi(x^{(P)})\|^2$) 是在 embedding 空间内的距离，而不是原始样本空间内的距离。

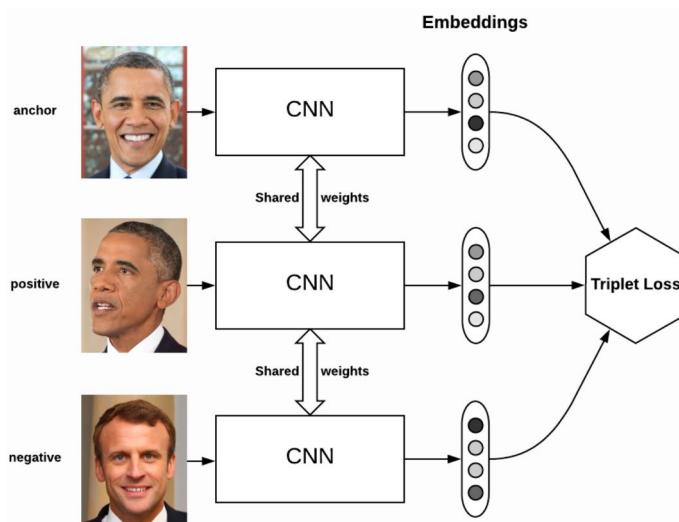


图 19: Triplet loss on two positive faces (Obama) and one negative face (Macron).

因此针对由三元组样本构成的数据集 $\mathcal{D} = \{(x^{(A)}, x^{(P)}, x^{(N)})\}$ 的训练目标函数为：

$$\min_{\phi} \mathcal{L}(\phi; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(x^{(A)}, x^{(P)}, x^{(N)}) \in \mathcal{D}} \max \left(d(f_{\phi}(x^{(A)}), f_{\phi}(x^{(P)})) - d(f_{\phi}(x^{(A)}), f_{\phi}(x^{(N)})) + \alpha, 0 \right)$$

注意：一定先对 embedding 进行归一化再学习，如此是针对 embedding 的方向学习而不是大小：

$$f_{\phi}(x^{(A)}) \leftarrow \frac{f_{\phi}(x^{(A)})}{\|f_{\phi}(x^{(A)})\|}, \quad f_{\phi}(x^{(P)}) \leftarrow \frac{f_{\phi}(x^{(P)})}{\|f_{\phi}(x^{(P)})\|}, \quad f_{\phi}(x^{(N)}) \leftarrow \frac{f_{\phi}(x^{(N)})}{\|f_{\phi}(x^{(N)})\|}$$

Triplet mining 对于一个三元组样本中的 Negative，一般可将一个三元组样本 $(x^{(A)}, x^{(P)}, x^{(N)})$ 分为三类：

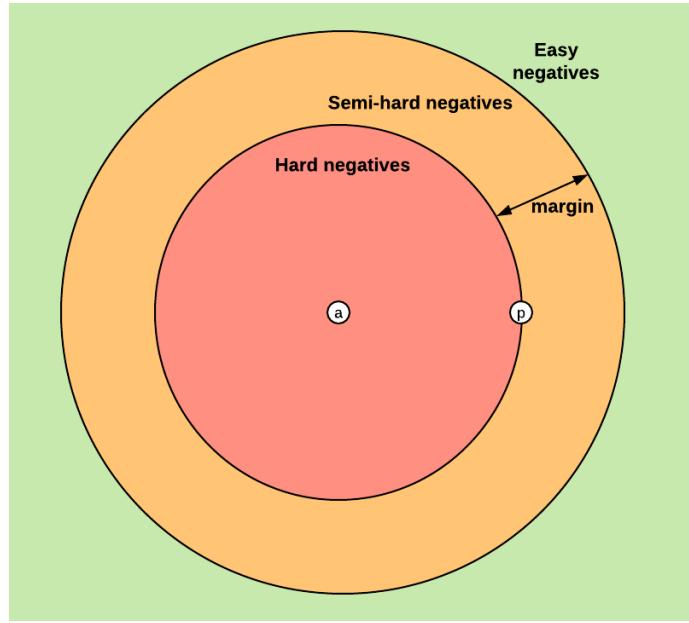


图 20: The three types of negatives, given an anchor and a positive. Image source: [Olivier Moindrot blog: Triplet Loss and Online Triplet Mining in TensorFlow](#).

1. **easy triplets (easy negatives):** triplets with loss of 0, because

$$d(f_{\phi}(x^{(A)}), f_{\phi}(x^{(P)})) + \alpha < d(f_{\phi}(x^{(A)}), f_{\phi}(x^{(N)}))$$

2. **semi-hard triplets (semi-hard negatives):** triplets where the negative is not closer to the anchor than the positive, but which still have positive loss:

$$d(f_{\phi}(x^{(A)}), f_{\phi}(x^{(P)})) < d(f_{\phi}(x^{(A)}), f_{\phi}(x^{(N)})) < d(f_{\phi}(x^{(A)}), f_{\phi}(x^{(P)})) + \alpha$$

3. **hard triplets (hard negatives):** triplets where the negative is closer to the anchor than the positive, i.e.

$$\begin{aligned} d(f_{\phi}(x^{(A)}), f_{\phi}(x^{(N)})) &< d(f_{\phi}(x^{(A)}), f_{\phi}(x^{(P)})) \\ \Rightarrow d(f_{\phi}(x^{(A)}), f_{\phi}(x^{(P)})) - d(f_{\phi}(x^{(A)}), f_{\phi}(x^{(N)})) &> 0 \end{aligned}$$

原始FaceNet[?] 论文中通过为每对 $(x^{(A)}, x^{(P)})$ 随机选择 semi-hard 的 $x^{(N)}$ ，构造出所有的三元组样本 $(x^{(A)}, x^{(P)}, x^{(N)})$ ，然后在这些三元组样本的集合 $\mathcal{D} = \{(x^{(A)}, x^{(P)}, x^{(N)})\}$ 上进行学习训练。

Offline and online triplet mining (未完成)

Offline triplet mining

Online triplet mining

Idea. 在深度模型中考虑使用层次化的多损失函数 (hierarchical multi loss functions)，譬如：

- 在 embedding (kernel) 层的学习中使用 triplet loss，更好地学习每个样本 x 对应的抽取特征 $f_\phi(x) \in \mathbb{R}^{D_{\text{kernel}}}$
- 在最后的 softmax 分类层的学习中使用 cross entropy loss，从而更好地学习分类系数 $W \in \mathbb{R}^{D_{\text{kernel}} \times K}$

而事实上已经有算法，特别是人脸识别领域的算法使用了这样 idea，用于同时加强 $f_\phi(x)$ 和 softmax 层参数 W 的多损失函数。譬如 Deep Learning Face Representation by Joint Identification-Verification (DeepID2)[?]

- Identification Loss:

$$\text{Ident}(f_\phi(x), t; W) = - \sum_{k=1}^K p_k \log \hat{y}_k = - \log \hat{p}_t \quad (\text{where } \hat{p}_t = \frac{e^{W_t^T f_\phi(x) + b_t}}{\sum_{k=1}^K e^{W_k^T f_\phi(x) + b_k}})$$

- Verification Loss:

$$\text{Verif}(f_\phi(x^{(i)}), f_\phi(x^{(j)}), y_{ij}, \phi) = \begin{cases} \frac{1}{2} \|f_\phi(x^{(i)}) - f_\phi(x^{(j)})\|_2^2 & \text{if: } y_{ij} = 1 \quad (\text{same person}) \\ \frac{1}{2} \max(0, m - \|f_\phi(x^{(i)}) - f_\phi(x^{(j)})\|_2)^2 & \text{if: } y_{ij} = -1 \quad (\text{different person}) \end{cases}$$

Reference

- Triplet Loss and Online Triplet Mining in TensorFlow
- In Defense of the Triplet Loss for Person Re-Identification[?]
- Brandon Amos: OpenFace 0.2.0: Higher accuracy and halved execution time
- Deep Learning Face Representations with Different Loss Functions for Face Recognition

3.3.13 COCO loss

论文出处：Learning Deep Features via Congenerous Cosine Loss for Person Recognition[?] 以及 Rethinking Feature Discrimination and Polymerization for Large-scale Recognition[?]

Reference

- Slides: Learning Deep Features via Congenerous Cosine Loss for Person Recognition(COCO loss)

3.3.14 Center loss

论文出处：[A Discriminative Feature Learning Approach for Deep Face Recognition\[?\]](#)

3.3.15 Center Invariant loss

论文出处：[Deep Face Recognition with Center Invariant Loss](#)

3.3.16 Wasserstein distance and the Kantorovich-Rubinstein duality

Earth Mover's Distance For discrete probability distributions, the Wasserstein distance is also descriptively called the earth mover's distance (EMD). If we imagine the distributions as different heaps of a certain amount of earth, then the EMD is the minimal total amount of work it takes to transform one heap into the other. **Work is defined as the amount of earth in a chunk times the distance it was moved.** Let's call our discrete distributions P_r and P_θ , each with l possible states(chunks) x or y respectively, and take two arbitrary distributions as an example.

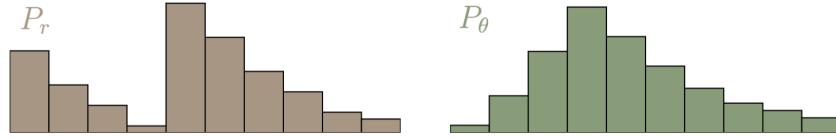


图 21: Probability distribution P_r and P_θ , each with ten states(chucks).

Calculating the EMD is in itself an optimization problem: There are infinitely many ways to move the earth around, and we need to find the optimal one. We call the transport plan that we are trying to find $\gamma(x, y) \in \mathbb{R}^{l \times l}$. It simply states how we distribute the amount of earth from one place y (one chunk in P_r) over the domain of x (one chunk in P_θ), or vice versa.

To be a valid transport plan, the transport plan $\gamma(x, y)$ must satisfy the following constraints:

$$\begin{cases} \sum_x \gamma(x, y) = P_r(y) & \forall y \in \{1, \dots, l\} \\ \sum_y \gamma(x, y) = P_\theta(x) & \forall x \in \{1, \dots, l\} \end{cases}$$

This ensures that following this plan yields the correct distributions.

Equivalently, we can call γ a **joined probability distribution** and require that:

$$\gamma \in \Pi(P_r, P_\theta)$$

where $\Pi(P_r, P_\theta)$ is the set of all distributions whose **marginal distributions** are P_r and P_θ respectively. To get the EMD, we have to multiply every value of $\gamma \in \mathbb{R}^{l \times l}$ with the Euclidian distance between x and y . With that, the definition of the Earth mover's distance is:

$$\begin{aligned} \text{EMD}(P_r, P_\theta) &= \inf_{\gamma \in \Pi} \sum_{x, y} \|x - y\| \gamma(x, y) \\ &= \inf_{\gamma \in \Pi} \mathbb{E}_{(x, y) \sim \gamma} \|x - y\| \end{aligned}$$

We can also set $\Gamma = \gamma(x, y)$ and $D = \|x - y\|$, with $\Gamma, D \in \mathbb{R}^{l \times l}$, Now we can write:

$$\text{EMD}(P_r, P_\theta) = \inf_{\gamma \in \Pi} \langle \Gamma, D \rangle_F$$

where $\langle \cdot, \cdot \rangle_F$ is the **Frobenius inner product** (sum of all the element-wise products).

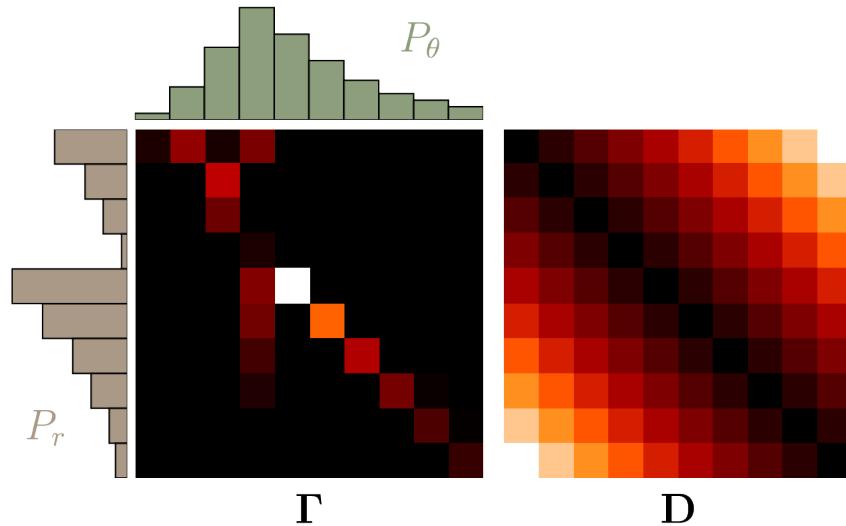


图 22: Transport plan Γ with marginal distributions P_r and P_θ , and distances D .

Linear Programming In the picture above, we can see the optimal transport plan Γ , It can be calculated using the generic method of Linear Programming(LP). With LP, we can solve problems of a certain canonical form:

- variable: Find a vector $\mathbf{x} \in \mathbb{R}^n$
- objective: that minimizes the cost $z = \mathbf{c}^\top \mathbf{x}$, $\mathbf{c} \in \mathbb{R}^n$
- constraint: \mathbf{x} is constrained by the equation $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{x} \geq \mathbf{0}$

To cast our problem of finding the EMD into this form, we have to flatten Γ and D as follow:

$$\begin{aligned}\mathbf{x} &= \text{vec}(\boldsymbol{\Gamma}) \in \mathbb{R}^{l^2} \\ \mathbf{c} &= \text{vec}(\mathbf{D}) \in \mathbb{R}^{l^2}\end{aligned}$$

and make RHS of constraint as follow:

$$\mathbf{b} = \begin{bmatrix} P_r \\ P_\theta \end{bmatrix} \in \mathbb{R}^{2l}$$

LHS of constraint as follow:

Reference

- vincentherrmann.github.io: Wasserstein GAN and the Kantorovich-Rubinstein Duality
- Read-through: Wasserstein GAN
- Wasserstein of Wasserstein Loss for Learning Generative Models[?]

3.4 Regularization

3.4.1 L1 regularization (Lasso regularization)

3.4.2 L2 regularization (Ridge regularization)

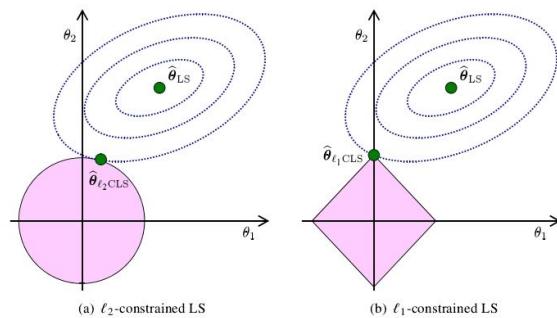


图 23: L1 (Lasso) regularization and L2 (Ridge) regularization.

3.4.3 Mini-batch aware regularization

在稀疏学习任务 (e.g. CTR 预估) 中, 为了防止过拟合, 需要引入正则化项, 但是又希望对参数 θ 的学习在每一个 mini-batch 内都能充分进行, 因此考虑对正则化项做一些 mini-batch 自适应的设计。

根据论文DIN, Zhou et al., 2018[?], 因此在一批训练数据 $\mathcal{B} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^M$ 内参数 w_j 的学习如下:

- 判断当前批训练数据 \mathcal{B} 内所有 sample 的第 j 维数据是否存在非 0 值:

$$I_j = \begin{cases} 1, & \exists (\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{B}, \text{ s.t. } x_j^{(i)} \neq 0 \\ 0, & \text{otherwise.} \end{cases}$$

- 设计属于当前批训练数据 \mathcal{B} 的正则化项:

$$\Omega_{\mathcal{B}}(w_j) = \lambda \frac{1}{n_j} w_j I_j, \quad (n_j \text{ 表示在所有训练样本 } \mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N \text{ 中第 } j \text{ 维数据非 0 的训练样本个数})$$

- 计算当前 mini-batch 数据 \mathcal{B} 作用下参数 w_j 的更新:

$$w_j \leftarrow w_j - \eta \left(\frac{1}{|\mathcal{B}|} \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{B}} \frac{\partial \mathcal{L}(f(\mathbf{x}^{(i)}; \theta), y^{(i)})}{\partial w_j} + \lambda \frac{1}{n_j} w_j I_j \right)$$

注意: 原始论文中正则化距离采用了 l1-norm, 也可以尝试 l2-norm 或者其他, 具体效果依照实验超参数调节结果而定。

因此, 自适应 (aware) 主要来自于两层次方面:

- 全部训练样本集 $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$ 层次: 维度 j 上非的样本出现次数越少, 惩罚越重, 有效防止过拟合 (原因: 维度 j 上非 0 的样本出现次数多了, 参数的有效学习几率更高, 学习更加充分, 因此几乎不存在因为有效训练样本太少而导致的过拟合问题, 所以惩罚反而更轻)。
- 批训练样本集 $\mathcal{B} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^M$ 层次: 批内数据中维度 j 上出现了非 0 值, 则作用惩罚项, 否则不作用惩罚项。

3.4.4 Summary of regularization

- 一视同仁的 regularization 策略 (e.g. L1 regularization, L2 regularization)
- 针对单个特征（考虑有效特征训练样本个数）的 regularization 策略 (e.g. Mini-batch aware regularization)
- 针对数值型和类别型特征，分别设置不同的 regularization 策略 ([未来可展开工作](#))
- 针对组 (group)、域 (field)、特征 (feature) 的重要程度，设置不同的 regularization 策略 ([未来可展开工作](#))

4 Optimization Algorithms

4.1 Overview

- 机器学习，深度学习以及强化学习的本质为解一个最优化问题
- 优化问题又可分为无约束优化问题 (unconstrained optimization) 和约束优化 (constrained optimization) 问题
- 求解优化问题就需要各种各样的优化算法，通常是对高度非凸优化问题进行优化求解

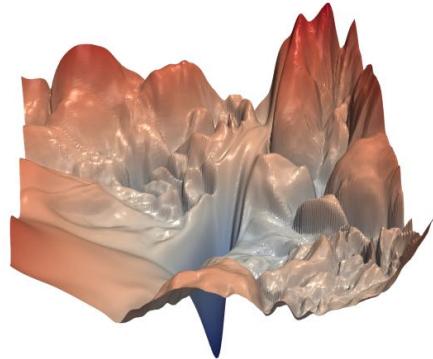


图 24: Visualizing the Loss Landscape of Neural Nets[?]

4.2 Gradient Descent Variants

每一次迭代更新参数时，我们会考虑如下内容：

- 参数迭代更新的准确性
- 参数迭代更新的速度

然后需要在这两者之间进行权衡。而批数据量 m 是决定这两者的因素，因此根据 m 大小不同，可以做如下分类：

- **Batch gradient descent:** 每次迭代使用全部训练数据更新参数

$$w \leftarrow w - \eta \cdot \frac{\partial J(\mathbf{w}; \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}; y^{(1)}, \dots, y^{(n)})}{\partial w}$$

- **Stochastic gradient descent:** 每次迭代使用随机选择的一条训练数据 $(\mathbf{x}^{(i)}, y^{(i)})$ 更新参数

$$w \leftarrow w - \eta \cdot \frac{\partial J(\mathbf{w}; \mathbf{x}^{(i)}; y^{(i)})}{\partial w}$$

- **Mini-batch gradient descent:** 每次迭代使用随机选择的 $m (m \ll n)$ 条训练数据更新参数

$$w \leftarrow w - \eta \cdot \frac{\partial J(\mathbf{w}; \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}; y^{(1)}, \dots, y^{(m)})}{\partial w}$$

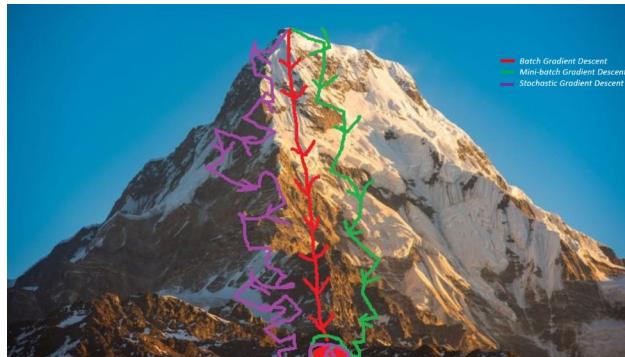


图 25: Comparison of gradient descent with different training samples

这三种方式各有特点：

- **Batch gradient descent:**

- 优点：更新最平稳
- 缺点：更新速度最慢

- **Stochastic gradient descent:**

- 优点：波动性会使参数跳跃到新的，潜在更好的局部最优点；更新速度最快
- 缺点：更新最不稳定，波动性太强；没有充分利用计算机的矩阵加速性能

- **Mini-batch gradient descent:** 二者的折中

- 优点 1：减少了参数更新时 SGD 的波动，做到更平稳的收敛
- 优点 2：充分利用深度学习库里高效的矩阵计算性能

关于 GD, SGD, MBGD 三者的形象对比：

- 多个“长者”，多个“人生经验”，少走“弯路”，走的稳
- “人生经验”太多记不下（空间开销），自行消化太费时（时间消耗）
- 随机选一部分“长者”的“人生经验”作为指导依据



图 26: Optimization task just like get down the mountain

4.3 Gradient Descent Optimization Algorithms

而基于 Mini-batch gradient descent，对学习率 η 和梯度方向 $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$ 进行适当的变化，可以得到如下一些梯度下降的变种优化算法 [?]。

4.3.1 SGD

Algorithm 1: Stochastic gradient descent (SGD) update[?]

Input: Learning rate η_0 ;

Initial parameter $\boldsymbol{\theta}$.

1 Initialize global step counter $t \leftarrow 0$

2 **while** stopping criterion not met **do**

3 Update global step counter $t \leftarrow t + 1$

4 Sample a minibatch of m samples from the training dataset with corresponding targets:

$$\{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^m.$$

5 Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$

6 Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_t \mathbf{g}$

7 **end**

算法解释

- SGD 算法一般还会对学习率 η 采用衰减策略，从而防止训练后期出现梯度爆炸的情况。

4.3.2 Momentum

Algorithm 2: Stochastic gradient descent with momentum[?]

Input: Learning rate η ;

Momentum parameter α ;

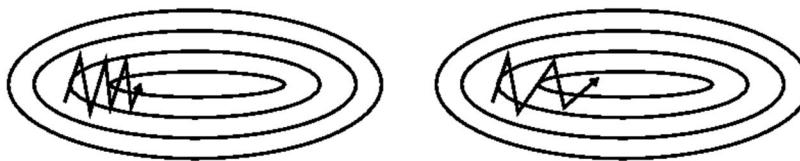
Initial parameter θ ;

Initial velocity v .

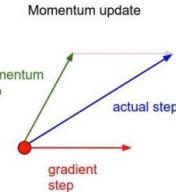
```

1 while stopping criterion not met do
2   Sample a minibatch of  $m$  samples from the training dataset with corresponding targets:
       $\{x^{(i)}, y^{(i)}\}_{i=1}^m$ .
3   Compute gradient estimate:  $\mathbf{g} \leftarrow \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} \mathcal{L}(f(\mathbf{x}^{(i)}; \theta), y^{(i)})$ 
4   Compute velocity update:  $\mathbf{v} \leftarrow \alpha \mathbf{v} - \eta \mathbf{g}$ 
5   Apply update:  $\theta \leftarrow \theta + \mathbf{v}$ 
6 end

```



(a) left—SGD without momentum, right—SGD with momentum



(b) Momentum update

图 27: Momentum algorithm and update

算法解释

- Momentum 优化算法相对于 SGD 考虑了先前迭代 step 的梯度：
 - 当前步更新方向 $-\mathbf{g}$ 和先前累计更新方向 \mathbf{v} 相似时，累计负梯度起到了加速并减小震荡的作用
 - 当前步更新方向 $-\mathbf{g}$ 和先前累计更新方向 \mathbf{v} 相反时，累计负梯度起到了减速并减小震荡的作用
- Momentum 相当于之前 $\frac{1}{1-\alpha}$ 个时刻的梯度向量和的平均值，因此最大可以实现 $\frac{1}{1-\alpha}$ 倍的加速
- α 一般取 0.9

4.3.3 Nesterov momentum

Algorithm 3: SGD with Nesterov momentum[?]

Input: Learning rate η ;

Momentum parameter α ;

Initial parameter θ ;

Initial velocity v .

```

1 while stopping criterion not met do
2   Sample a minibatch of  $m$  samples from the training dataset with corresponding targets:
3      $\{x^{(i)}, y^{(i)}\}_{i=1}^m$ .
4   Apply interim update:  $\tilde{\theta} \leftarrow \theta + \alpha v$ 
5   Compute gradient (at interim point):  $g \leftarrow \frac{1}{m} \sum_{i=1}^m \nabla_{\tilde{\theta}} \mathcal{L}(f(x^{(i)}; \tilde{\theta}), y^{(i)})$ 
6   Compute velocity update:  $v \leftarrow \alpha v - \eta g$ 
7   Apply update:  $\theta \leftarrow \theta + v$ 
8 end

```

算法解释

- Nesterov momentum 优化算法最核心的思想就是**预判前方走势**
- 因为步骤 $\tilde{\theta} \leftarrow \theta + \alpha v$ 通过在累计方向 v 上前进一小步，将优化初始点带到了新的出发点 $\tilde{\theta}$
- 从当前迭代的原始出发点 θ 到新出发点 $\tilde{\theta}$ 的方向和累计更新方向 v 是相同的，即“**顺应大势所趋，在潮流方向上先偷偷挪了一小步**”
- 所以相对于 Momentum 算法，更可以实现加速的目的。

SGD, Momentum, Nesterov 优化算法存在的问题

- 在每一个 step 的更新中，找到一个合适的学习率 η 是十分困难的。而学习率 η 太小导致收敛速度太慢，学习率 η 太大又会导致在损失函数最小值点附近波动甚至发散。

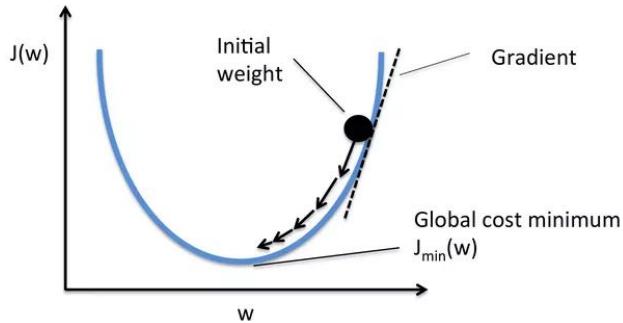


图 28: Loss function & gradient based optimization

- 训练过程中，学习率的衰减策略 η 是人为制定的，不能够自动匹配数据的特征
- 所有参数的更新都使用相同的学习率 η 。如果数据非常稀疏，并且特征的频率大为不同，我们不应该对所有的特征使用相同的学习率 η ，而应该对出现频率低的特征使用相对大的学习率，而对出现频率高的特征使用相对小的学习率
- 很难避免次优点（鞍点 saddle points）

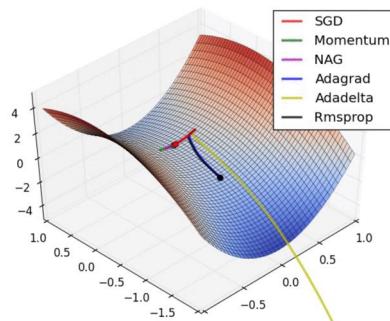


图 29: Hard for SGD, Momentum, Nesterov optimizers to avoid saddle point

解决办法：使用**自适应学习率优化算法** (Adaptive learning rate)

4.3.4 AdaGrad

Algorithm 4: The AdaGrad algorithm[?]

Input: Global learning rate η ;

Initial parameter θ ;

Small constant ϵ , perhaps 10^{-7} for numerical stability.

1 Initialize gradient accumulation variable $r = \mathbf{0}$

2 **while** stopping criterion not met **do**

3 Sample a minibatch of m samples from the training dataset with corresponding targets:

$$\{x^{(i)}, y^{(i)}\}_{i=1}^m.$$

4 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} \mathcal{L}(f(x^{(i)}; \theta), y^{(i)})$

5 Accumulate squared gradient: $r \leftarrow r + \mathbf{g} \odot \mathbf{g}$

6 Compute update: $\Delta\theta \leftarrow -\frac{\eta}{\sqrt{r} + \epsilon} \odot \mathbf{g}$ (Division and square root applied element-wise)

7 Apply update: $\theta \leftarrow \theta + \Delta\theta$

8 **end**

算法解释

- **算法优点：**特别适合稀疏数据 (sparse data)，因为每一步迭代更新的过程中，调整了学习率，使得不频繁的参数 (infrequent parameter) 得到更大的更新，频繁的参数 (frequent parameter) 得到更小的更新。

– **公式解释** 以 Logistic Regression 为例，迭代更新公式 (为了方便不考虑正则化项) 如下：

$$w_j \leftarrow w_j + \eta \left(\frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)} \right)$$

当数据集的第 j 个维度总为 0 (稀疏) 时， $g_j = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$ 的取值非常小，所以为了让参数 w_j 得到足够充分的更新，需要学习率 η 非常大，但此时别的特征维度上数据稠密的话， $g_j = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$ 的取值就非常大，因此会发散！

– **解决办法** Adagrad 的解决办法是独立地累积每一个维度上梯度的平方，在第 t 个 step 任意维度 j 上的梯度平方累积为：

$$r_j = \sum_{t'=1}^t g_{j,t'}^2$$

* 维度 j 稀疏：

$$g_{j,t'}^2 \text{ 小} \Rightarrow r_j = \sum_{t'=1}^t g_{j,t'}^2 \text{ 小} \Rightarrow \sqrt{r_j} + \epsilon \text{ 小} \Rightarrow \underbrace{\frac{\eta}{\sqrt{r_j} + \epsilon}}_{\text{实际学习率}} \text{ 大}$$

* 维度 j 稠密：

$$g_{j,t'}^2 \text{ 大} \Rightarrow r_j = \sum_{t'=1}^t g_{j,t'}^2 \text{ 大} \Rightarrow \sqrt{r_j} + \epsilon \text{ 大} \Rightarrow \underbrace{\frac{\eta}{\sqrt{r_j} + \epsilon}}_{\text{实际学习率}} \text{ 小}$$

所以实现了学习率的自适应调节。

- **算法缺点：**训练中后期学习率为 0，训不动了，或者直接使得训练提前结束。因此针对这个问题，后期又有了更好的优化算法（AdaDelta, RMSProp, Adam…）。

4.3.5 AdaDelta

Algorithm 5: The Adadelta algorithm[?]

Input: Global learning rate η ;

Decay rate ρ ;

Initial parameter θ_0 ;

Small constant ϵ , usually 10^{-8} used to stabilize division by small numbers.

- 1 Initialize global step counter $t \leftarrow 0$
 - 2 Initialize squared gradient accumulation variable $r_t = \mathbf{0}$
 - 3 Initialize squared update of parameter accumulation variable $s_t = \mathbf{0}$
 - 4 **while** stopping criterion not met **do**
 - 5 Update global step counter $t \leftarrow t + 1$
 - 6 Sample a minibatch of m samples from the training dataset with corresponding targets:
 $\{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^m$.
 - 7 Compute gradient: $\mathbf{g}_t \leftarrow \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} \mathcal{L}(f(\mathbf{x}^{(i)}; \theta), y^{(i)})$
 - 8 Accumulate squared gradient: $\mathbf{r}_t \leftarrow \rho \mathbf{r}_{t-1} + (1 - \rho) \mathbf{g}_t \odot \mathbf{g}_t$
 - 9 Compute update: $\Delta\theta_t \leftarrow -\eta \frac{\sqrt{s_{t-1} + \epsilon}}{\sqrt{\mathbf{r}_t + \epsilon}} \odot \mathbf{g}_t$ (Division and square root applied element-wise)
 - 10 Accumulate update: $\mathbf{s}_t \leftarrow \rho \mathbf{s}_{t-1} + (1 - \rho) \Delta\theta_t \odot \Delta\theta_t$
 - 11 Apply update: $\theta_{t+1} \leftarrow \theta_t + \Delta\theta_t$
 - 12 **end**
-

算法解释

- Adadelta 算法中 Global learning rate η 实质为一个 smoothing factor.
- 当先前累计的参数更新量大时，当前 step 的参数更新值就大。
- 当先前累计的梯度大时（说明可学习样本充分），当前 step 的参数更新值就小。

4.3.6 RMSProp

Algorithm 6: The RMSProp algorithm[?]

Input: Global learning rate η , decay rate ρ ;

Initial parameter θ ;

Small constant ϵ , usually 10^{-10} used to stabilize division by small numbers.

1 Initialize squared gradient accumulation variable $r = \mathbf{0}$

2 **while** stopping criterion not met **do**

3 Sample a minibatch of m samples from the training dataset with corresponding targets:

$$\{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^m.$$

4 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} \mathcal{L}(f(\mathbf{x}^{(i)}; \theta), y^{(i)})$

5 Accumulate squared gradient: $r \leftarrow \rho r + (1 - \rho) \mathbf{g} \odot \mathbf{g}$

6 Compute update: $\Delta\theta \leftarrow -\frac{\eta}{\sqrt{r + \epsilon}} \odot \mathbf{g}$ (Division and square root applied element-wise)

7 Apply update: $\theta \leftarrow \theta + \Delta\theta$

8 **end**

算法解释

- 只累计之前最新的 $\frac{1}{1-\rho}$ 个梯度的平方，因此 r_j 不会一直增大，同时同一维度在时间 step 上实现学习率的自适应调节。
- RMSProp 算法相对于 AdaGrad 算法，其将二阶梯度累积量由：

$$r \leftarrow r + g \odot g$$

替换为：

$$r \leftarrow \rho r + (1 - \rho) g \odot g$$

4.3.7 Adam

Algorithm 7: The Adam algorithm[?]

Input: Global learning rate η (Suggested default: 0.001);
 Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1]$ (Suggested defaults: 0.9 and 0.999 respectively);
 Small constant ϵ used for numerical stabilization (Suggested default: 10^{-8}); Initial parameter θ .

```

1 Initialize 1st and 2nd moment variables:  $s = \mathbf{0}$ ,  $r = \mathbf{0}$ 
2 Initialize time step  $t = 0$ 
3 while stopping criterion not met do
4     Sample a minibatch of  $m$  samples with corresponding targets:  $\{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^m$ .
5     Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} \mathcal{L}(f(\mathbf{x}^{(i)}; \theta), y^{(i)})$ 
6      $t \leftarrow t + 1$ 
7     Update biased first moment estimate:  $s \leftarrow \rho_1 s + (1 - \rho_1) \mathbf{g}$ 
8     Update biased second moment estimate:  $r \leftarrow \rho_2 r + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$ 
9     Correct bias in first moment:  $\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$ 
10    Correct bias in second moment:  $\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}$ 
11    Compute update:  $\Delta\theta = -\eta \frac{\hat{s}}{\sqrt{\hat{r}} + \epsilon}$  (operations applied element-wise)
12    Apply update:  $\theta \leftarrow \theta + \Delta\theta$ 
13 end

```

算法解释

- 引入一阶动量是为了调整优化方向。
- 引入二阶动量是为了调整优化步长。
- 把一阶动量和二阶动量都用起来，同时做了修正，那这就是 Adam(adaptive + momentum) 算法了。

TensorFlow Implement

```

# Copyright (C) 2020 Tong Jia. All rights reserved.
from tensorflow.python.eager import context
from tensorflow.python.framework import ops
from tensorflow.python.ops import control_flow_ops
from tensorflow.python.ops import math_ops
from tensorflow.python.ops import resource_variable_ops
from tensorflow.python.ops import state_ops
from tensorflow.python.training import optimizer
from tensorflow.python.training import training_ops

class AdamOptimizer(optimizer.Optimizer):
    """Optimizer that implements the Adam algorithm.

```

References:

Adam - A Method for Stochastic Optimization:
 [Kingma et al., 2015] (<https://arxiv.org/abs/1412.6980>)
 ([pdf] (<https://arxiv.org/pdf/1412.6980.pdf>))

'''

```
def __init__(self,
            learning_rate=0.001,
            beta1=0.9,
            beta2=0.999,
            epsilon=1e-8,
            use_locking=False,
            name="Adam"):

    """Construct a new Adam optimizer.

    Initialization:
    $m_0 := 0 \text{(Initialize initial 1st moment vector)}$$
    $$v_0 := 0 \text{(Initialize initial 2nd moment vector)}$$
    $$t := 0 \text{(Initialize timestep)}$$

    The update rule for `variable` with gradient `g` uses an optimization
    described at the end of section 2 of the paper:
    $$t := t + 1$$
    $$\text{lr}_t := \text{learning\_rate} * \sqrt{1 - \beta_2^t} / (1 - \beta_1^t)$$
    $$m_t := \beta_1 * m_{t-1} + (1 - \beta_1) * g$$
    $$v_t := \beta_2 * v_{t-1} + (1 - \beta_2) * g * g$$
    $$\text{variable} := \text{variable} - \text{lr}_t * m_t / (\sqrt{v_t} + \epsilon)$$

    The default value of  $1e-8$  for epsilon might not be a good default in
    general. For example, when training an Inception network on ImageNet a
    current good choice is 1.0 or 0.1. Note that since AdamOptimizer uses the
    formulation just before Section 2.1 of the Kingma and Ba paper rather than
    the formulation in Algorithm 1, the "epsilon" referred to here is "epsilon
    hat" in the paper.

    The sparse implementation of this algorithm (used when the gradient is an
    IndexedSlices object, typically because of `tf.gather` or an embedding
    lookup in the forward pass) does apply momentum to variable slices even if
    they were not used in the forward pass (meaning they have a gradient equal
    to zero). Momentum decay (beta1) is also applied to the entire momentum
    accumulator. This means that the sparse behavior is equivalent to the dense
    behavior (in contrast to some momentum implementations which ignore momentum
    unless a variable slice was actually used).

    Args:
        learning_rate: A Tensor or a floating point value. The learning rate.
        beta1: A float value or a constant float tensor. The exponential decay
               rate for the 1st moment estimates.
        beta2: A float value or a constant float tensor. The exponential decay
               rate for the 2nd moment estimates.
        epsilon: A small constant for numerical stability. This epsilon is
                 "epsilon hat" in the Kingma and Ba paper (in the formula just before
                 Section 2.1), not the epsilon in Algorithm 1 of the paper.
        use_locking: If True use locks for update operations.
        name: Optional name for the operations created when applying gradients.
```

```

    Defaults to "Adam". @compatibility(eager) When eager execution is
    enabled, `learning_rate`, `beta1`, `beta2`, and `epsilon` can each be a
    callable that takes no arguments and returns the actual value to use.
    This can be useful for changing these values across different
    invocations of optimizer functions. @end_compatibility
"""

super(AdamOptimizer, self).__init__(use_locking, name)
self._lr = learning_rate
self._beta1 = beta1
self._beta2 = beta2
self._epsilon = epsilon

# Tensor versions of the constructor arguments, created in _prepare().
self._lr_t = None
self._beta1_t = None
self._beta2_t = None
self._epsilon_t = None

def _get_beta_accumulators(self):
    with ops.init_scope():
        if context.executing_eagerly():
            graph = None
        else:
            graph = ops.get_default_graph()
    return (self._get_non_slot_variable("beta1_power", graph=graph),
            self._get_non_slot_variable("beta2_power", graph=graph))

def _create_slots(self, var_list):
    # Create the beta1 and beta2 accumulators on the same device as the first
    # variable. Sort the var_list to make sure this device is consistent across
    # workers (these need to go on the same PS, otherwise some updates are
    # silently ignored).
    first_var = min(var_list, key=lambda x: x.name)
    self._create_non_slot_variable(
        initial_value=self._beta1, name="beta1_power", colocate_with=first_var)
    self._create_non_slot_variable(
        initial_value=self._beta2, name="beta2_power", colocate_with=first_var)

    # Create slots for the first and second moments.
    for v in var_list:
        self._zeros_slot(v, "m", self._name)
        self._zeros_slot(v, "v", self._name)

def _prepare(self):
    lr = self._call_if_callable(self._lr)
    beta1 = self._call_if_callable(self._beta1)
    beta2 = self._call_if_callable(self._beta2)
    epsilon = self._call_if_callable(self._epsilon)

    self._lr_t = ops.convert_to_tensor(lr, name="learning_rate")
    self._beta1_t = ops.convert_to_tensor(beta1, name="beta1")
    self._beta2_t = ops.convert_to_tensor(beta2, name="beta2")

```

```

    self._epsilon_t = ops.convert_to_tensor(epsilon, name="epsilon")

    def _apply_dense(self, grad, var):
        m = self.get_slot(var, "m")
        v = self.get_slot(var, "v")
        beta1_power, beta2_power = self._get_beta_accumulators()
        return training_ops.apply_adam(
            var,
            m,
            v,
            math_ops.cast(beta1_power, var.dtype.base_dtype),
            math_ops.cast(beta2_power, var.dtype.base_dtype),
            math_ops.cast(self._lr_t, var.dtype.base_dtype),
            math_ops.cast(self._beta1_t, var.dtype.base_dtype),
            math_ops.cast(self._beta2_t, var.dtype.base_dtype),
            math_ops.cast(self._epsilon_t, var.dtype.base_dtype),
            grad,
            use_locking=self._use_locking).op

    def _resource_apply_dense(self, grad, var):
        m = self.get_slot(var, "m")
        v = self.get_slot(var, "v")
        beta1_power, beta2_power = self._get_beta_accumulators()
        return training_ops.resource_apply_adam(
            var.handle,
            m.handle,
            v.handle,
            math_ops.cast(beta1_power, grad.dtype.base_dtype),
            math_ops.cast(beta2_power, grad.dtype.base_dtype),
            math_ops.cast(self._lr_t, grad.dtype.base_dtype),
            math_ops.cast(self._beta1_t, grad.dtype.base_dtype),
            math_ops.cast(self._beta2_t, grad.dtype.base_dtype),
            math_ops.cast(self._epsilon_t, grad.dtype.base_dtype),
            grad,
            use_locking=self._use_locking)

    def _apply_sparse_shared(self, grad, var, indices, scatter_add):
        beta1_power, beta2_power = self._get_beta_accumulators()
        beta1_power = math_ops.cast(beta1_power, var.dtype.base_dtype)
        beta2_power = math_ops.cast(beta2_power, var.dtype.base_dtype)
        lr_t = math_ops.cast(self._lr_t, var.dtype.base_dtype)
        beta1_t = math_ops.cast(self._beta1_t, var.dtype.base_dtype)
        beta2_t = math_ops.cast(self._beta2_t, var.dtype.base_dtype)
        epsilon_t = math_ops.cast(self._epsilon_t, var.dtype.base_dtype)
        lr = (lr_t * math_ops.sqrt(1 - beta2_power) / (1 - beta1_power))
        # m_t = beta1 * m + (1 - beta1) * g_t
        m = self.get_slot(var, "m")
        m_scaled_g_values = grad * (1 - beta1_t)
        m_t = state_ops.assign(m, m * beta1_t, use_locking=self._use_locking)
        with ops.control_dependencies([m_t]):
            m_t = scatter_add(m, indices, m_scaled_g_values)
        # v_t = beta2 * v + (1 - beta2) * (g_t * g_t)

```

```

v = self.get_slot(var, "v")
v_scaled_g_values = (grad * grad) * (1 - beta2_t)
v_t = state_ops.assign(v, v * beta2_t, use_locking=self._use_locking)
with ops.control_dependencies([v_t]):
    v_t = scatter_add(v, indices, v_scaled_g_values)
v_sqrt = math_ops.sqrt(v_t)
var_update = state_ops.assign_sub(
    var, lr * m_t / (v_sqrt + epsilon_t), use_locking=self._use_locking)
return control_flow_ops.group(*[var_update, m_t, v_t])

def _apply_sparse(self, grad, var):
    return self._apply_sparse_shared(
        grad.values,
        var,
        grad.indices,
        lambda x, i, v: state_ops.scatter_add( # pylint: disable=g-long-lambda
            x,
            i,
            v,
            use_locking=self._use_locking))

def _resource_scatter_add(self, x, i, v):
    with ops.control_dependencies(
        [resource_variable_ops.resource_scatter_add(x.handle, i, v)]):
        return x.value()

def _resource_apply_sparse(self, grad, var, indices):
    return self._apply_sparse_shared(grad, var, indices,
                                    self._resource_scatter_add)

def _finish(self, update_ops, name_scope):
    # Update the power accumulators.
    with ops.control_dependencies(update_ops):
        beta1_power, beta2_power = self._get_beta_accumulators()
        with ops.colocate_with(beta1_power):
            update_beta1 = beta1_power.assign(
                beta1_power * self._beta1_t, use_locking=self._use_locking)
            update_beta2 = beta2_power.assign(
                beta2_power * self._beta2_t, use_locking=self._use_locking)
    return control_flow_ops.group(
        *update_ops + [update_beta1, update_beta2], name=name_scope)

```

Torch Implement

```

# Copyright (C) 2020 Tong Jia. All rights reserved.
import math
import torch
from torch.optim.optimizer import Optimizer

```

```

class Adam(Optimizer):
    """Implements Adam algorithm.
    It has been proposed in `Adam: A Method for Stochastic Optimization`_.
    Arguments:
        params (iterable): iterable of parameters to optimize or dicts defining
            parameter groups
        lr (float, optional): learning rate (default: 1e-3)
        betas (Tuple[float, float], optional): coefficients used for computing
            running averages of gradient and its square (default: (0.9, 0.999))
        eps (float, optional): term added to the denominator to improve
            numerical stability (default: 1e-8)
        weight_decay (float, optional): weight decay (L2 penalty) (default: 0)
        amsgrad (boolean, optional): whether to use the AMSGrad variant of this
            algorithm from the paper `On the Convergence of Adam and Beyond`_
            (default: False)
    .. _Adam\: A Method for Stochastic Optimization:
        https://arxiv.org/abs/1412.6980
    .. _On the Convergence of Adam and Beyond:
        https://openreview.net/forum?id=ryQu7f-RZ
    """
    def __init__(self, params, lr=1e-3, betas=(0.9, 0.999), eps=1e-8,
                 weight_decay=0, amsgrad=False):
        if not 0.0 <=lr:
            raise ValueError("Invalid learning rate: {}".format(lr))
        if not 0.0 <=eps:
            raise ValueError("Invalid epsilon value: {}".format(eps))
        if not 0.0 <=betas[0] <1.0:
            raise ValueError("Invalid beta parameter at index 0: {}".format(betas[0]))
        if not 0.0 <=betas[1] <1.0:
            raise ValueError("Invalid beta parameter at index 1: {}".format(betas[1]))
        defaults =dict(lr=lr, betas=betas, eps=eps,
                       weight_decay=weight_decay, amsgrad=amsgrad)
        super(Adam, self).__init__(params, defaults)

    def __setstate__(self, state):
        super(Adam, self).__setstate__(state)
        for group in self.param_groups:
            group.setdefault('amsgrad', False)

    def step(self, closure=None):
        """Performs a single optimization step.
        Arguments:
            closure (callable, optional): A closure that reevaluates the model
                and returns the loss.
        """
        loss =None
        if closure is not None:
            loss =closure()

        for group in self.param_groups:
            for p in group['params']:

```

```

if p.grad is None:
    continue
grad = p.grad.data
if grad.is_sparse:
    raise RuntimeError('Adam does not support sparse gradients, please consider SparseAdam
                           instead')
amsgrad = group['amsgrad']

state = self.state[p]

# State initialization
if len(state) == 0:
    state['step'] = 0
    # Exponential moving average of gradient values
    state['exp_avg'] = torch.zeros_like(p.data, memory_format=torch.preserve_format)
    # Exponential moving average of squared gradient values
    state['exp_avg_sq'] = torch.zeros_like(p.data, memory_format=torch.preserve_format)
if amsgrad:
    # Maintains max of all exp. moving avg. of sq. grad. values
    state['max_exp_avg_sq'] = torch.zeros_like(p.data, memory_format=torch.preserve_format)

exp_avg, exp_avg_sq = state['exp_avg'], state['exp_avg_sq']
if amsgrad:
    max_exp_avg_sq = state['max_exp_avg_sq']
beta1, beta2 = group['betas']

state['step'] += 1
bias_correction1 = 1 - beta1 ** state['step']
bias_correction2 = 1 - beta2 ** state['step']

if group['weight_decay'] != 0:
    grad = grad.add(group['weight_decay'], p.data)

# Decay the first and second moment running average coefficient
exp_avg.mul_(beta1).add_(1 - beta1, grad)
exp_avg_sq.mul_(beta2).addcmul_(1 - beta2, grad, grad)
if amsgrad:
    # Maintains the maximum of all 2nd moment running avg. till now
    torch.max(max_exp_avg_sq, exp_avg_sq, out=max_exp_avg_sq)
    # Use the max. for normalizing running avg. of gradient
    denom = (max_exp_avg_sq.sqrt() / math.sqrt(bias_correction2)).add_(group['eps'])
else:
    denom = (exp_avg_sq.sqrt() / math.sqrt(bias_correction2)).add_(group['eps'])

step_size = group['lr'] / bias_correction1

p.data.addcdiv_(-step_size, exp_avg, denom)

return loss

```

4.3.8 AdaBound

Algorithm 8: The AdaBound algorithm[?]

Input: Global learning rate α (Suggested default: 0.001);
 Lower bound function for learning rate clipping $\eta_l(t)$ (t is the time step);
 Upper bound function for learning rate clipping $\eta_u(t)$;
 Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1]$ (Suggested defaults: 0.9 and 0.999 respectively);
 Small constant ϵ used for numerical stabilization (Suggested default: 10^{-8}); Initial parameter θ .

```

1 Initialize 1st and 2nd moment variables:  $s = \mathbf{0}$ ,  $r = \mathbf{0}$ 
2 Initialize time step  $t = 0$ 
3 while stopping criterion not met do
4   Sample a minibatch of  $m$  samples with corresponding targets:  $\{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^m$ .
5   Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} L(f(\mathbf{x}^{(i)}; \theta), y^{(i)})$ 
6    $t \leftarrow t + 1$ 
7   Update biased first moment estimate:  $s \leftarrow \rho_1 s + (1 - \rho_1) \mathbf{g}$ 
8   Update biased second moment estimate:  $r \leftarrow \rho_2 r + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$ 
9   Clip learning rate  $\frac{\alpha}{\sqrt{r + \epsilon}}$  by  $\eta_l(t)$  and  $\eta_u(t)$  and correct it:
    
$$\hat{\eta} \leftarrow \text{Clip}\left(\frac{\alpha}{\sqrt{r + \epsilon}}, \eta_l(t), \eta_u(t)\right), \quad \eta \leftarrow \frac{\hat{\eta}}{\sqrt{t}}$$

10  Compute update:  $\Delta\theta = -\eta \odot s$  (operations applied element-wise)
11  Apply update:  $\theta \leftarrow \theta + \Delta\theta$ 
12 end
```

算法解释

- AdaBound 的核心在于 **学习率剪裁 (learning rate clipping)**
- Adam 优化算法虽然比 SGD 更快，但存在两个重大缺陷：
 - 结果不收敛
 - 找不到全局最优点

而造成这两大缺陷的原因为：

- 不稳定的学习率 (e.g. 某一维度上一个 time step 为 1000，下一个 time step 为 0.1)
- 极端学习率 (e.g. 某一个 time step 时，一个维度为 1000，另一个维度为 0.1)

因此 AdaBound 解决这个问题的方式是将任意时刻 t 的学习率限制在 $\eta_l(t)$ 和 $\eta_u(t)$ 之间

- 因此希望优化算法在训练前期像 Adam 一样快速，后期像 SGD 一样效果好，而：
 - 当 $\eta_l(t) = \alpha$ 且 $\eta_u(t) = \alpha$ 时：优化算法即为 SGD

- 当 $\eta_l(t) = 0$ 且 $\eta_u(t) = \infty$ 时：优化算法即为 Adam

因此为了实现算法从 Adam 到 SGD 的平滑过度，则将 $\eta_l(t)$ 和 $\eta_u(t)$ 变为随时间变化的函数：

- lower bound $\eta_l(t)$ ：从 0 到 α 的递增
- upper bound $\eta_u(t)$ ：从 ∞ 到 α 的递减

在这种情况下，AdaBound 开始时像 Adam 一样训练速度很快，随着学习率边界越来越受到限制，它又逐渐转变为 SGD

- $\eta_l(t)$ 从 0 到 α 的具体递增策略，以及 $\eta_u(t)$ 从 ∞ 到 α 的具体递减策略可以人为设置
- AdaBound 的优点：对超参数（e.g. global learning rate α ）不是很敏感，省去了大量调参的时间

4.3.9 RAdam (未完成)

Algorithm 9: Rectified Adam. All operations are element-wise[?].

Input: Learning rate η ;
Momentum parameter α ;
Initial parameter θ ;
Initial velocity v .

1 **while** stopping criterion not met **do**
2 **end**

4.3.10 SWATS: Improving Generalization Performance by Switching from Adam to SGD

Algorithm 10: SWATS[?]

Input: Objective function L ;

Initial parameter θ_0 ;

Learning rate $\eta = 10^{-3}$, accumulator coefficients $(\beta_1, \beta_2) = (0.9, 0.999)$;

Small constant $\epsilon = 10^{-9}$ used to stabilize division by small numbers, phase = Adam.

1 Initialize global step counter $t \leftarrow 0$, velocity for SGDM $v_t \leftarrow \mathbf{0}$, 1st and 2nd variables for Adam $m_t \leftarrow \mathbf{0}$, $a_t \leftarrow \mathbf{0}$, exponential average value $\lambda_t \leftarrow 0$.

2 **while** stopping criterion not met **do**

3 Update global step counter: $t = t + 1$

4 Sample a minibatch of m samples from the training dataset with corresponding targets:
 $\{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^m$.

5 Compute gradients: $\mathbf{g}_t \leftarrow \frac{1}{m} \nabla_{\theta} \sum_{i=1}^m L(f(\mathbf{x}^{(i)}; \theta_{t-1}), y^{(i)})$

6 **if** $phase = SGD$ **then**

7 Compute velocity update: $\mathbf{v}_t = \beta_1 \mathbf{v}_{t-1} - \mathbf{g}_t$ (Note there doesn't exist η in front of \mathbf{g}_t)

8 Update parameter: $\theta_t = \theta_{t-1} + (1 - \beta_1) \Lambda \mathbf{v}_t$

9 **continue**

10 **end**

11 Update 1st variables for Adam: $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$

12 Update 2nd variables for Adam: $\mathbf{a}_t = \beta_2 \mathbf{a}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$

13 Correct bias of 1st and 2nd variables for Adam: $\widehat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}$

14 Correct bias of 2nd variables for Adam: $\widehat{\mathbf{a}}_t = \frac{\mathbf{a}_t}{1 - \beta_2^t}$

15 Compute update of Adam: $\mathbf{p}_t = -\eta \frac{\widehat{\mathbf{m}}_t}{\sqrt{\widehat{\mathbf{a}}_t + \epsilon}}$ (element-wise op)

16 **if** $\mathbf{p}_t^T \mathbf{g}_t \neq 0$ **then**

17 $\gamma_t = \frac{\mathbf{p}_t^T \mathbf{p}_t}{-\mathbf{p}_t^T \mathbf{g}_t} \in \mathbb{R}$, $\lambda_t = \beta_2 \lambda_{t-1} + (1 - \beta_2) \gamma_t$

18 **if** $t > 1$ and $|\frac{\lambda_t}{(1 - \beta_2^t)} - \gamma_t| < \epsilon$ **then**

19 phase = SGD

20 $\mathbf{v}_t = \mathbf{0}$, $\Lambda = \frac{\lambda_t}{1 - \beta_2^t} \in \mathbb{R}$

21 **end**

22 **else**

23 $\lambda_t = \lambda_{t-1}$

24 **end**

25 **end**

算法解释 论文表述了存在的两个问题：

- **Problem-1:**

何时切换优化算法 Adam \Rightarrow SGD ? 疑难点在于：切换太晚，Adam 可能已经跑到自己的盆地去了，SGD 再怎么好也跑不出来了；切换太早，影响收敛速度和效果。

- **Problem-2:**

Adam \Rightarrow SGD 切换算法后使用什么样的学习率 η^{SGD} ? 疑难点在于: Adam 使用自适应学习率, 依赖的是二阶动量的积累, SGD 接着训练的话, 很难找到一个很好的衔接学习率。

解答如下:

- Solution-1:

- Solution-2:

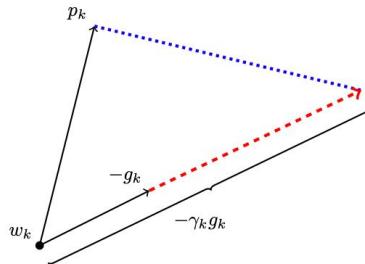


图 30: Illustrating the learning rate for SGD (γ_t) estimated by our proposed projection given an iterate θ_t , a stochastic gradient g_t and the Adam step p_t

$$\text{– Adam 的更新方向: } \theta_t^{\text{Adam}} = -\frac{\eta \hat{m}_t}{\sqrt{\hat{a}_t} + \epsilon}$$

$$\text{– SGD 的更新方向: } \theta_t^{\text{SGD}} = -\eta^{\text{SGD}} g_t$$

如果 SGD 要接过 Adam 的大旗, 那么需要先后做两件事情:

- 首先沿着 Adam 的更新方向 θ_t^{Adam} 走一步老路
- 而后沿着 θ_t^{Adam} 的正交方向走一步属于自己的新路

那么用数学语言表示, 即:

$$\text{– 将 } \theta_t^{\text{SGD}} \text{ 分解为 } \theta_t^{\text{Adam}} \text{ 方向和其正交方向两个方向}$$

$$\text{– } \theta_t^{\text{SGD}} \text{ 在 } \theta_t^{\text{Adam}} \text{ 方向上的投影, 应该正好等于 } \theta_t^{\text{Adam}}$$

方程描述如下:

$$\begin{aligned} \text{proj } \theta_t^{\text{SGD}} &= \theta_t^{\text{Adam}} \\ \Rightarrow \frac{(\theta_t^{\text{SGD}})^T \odot \theta_t^{\text{Adam}}}{|\theta_t^{\text{Adam}}|} &= \theta_t^{\text{Adam}} \\ \Rightarrow \frac{((\theta_t^{\text{SGD}})^T \odot \theta_t^{\text{Adam}})^T ((\theta_t^{\text{SGD}})^T \odot \theta_t^{\text{Adam}})}{(\theta_t^{\text{Adam}})^T \theta_t^{\text{Adam}}} & \end{aligned}$$

4.4 Online Learning Optimization Algorithms

4.4.1 FTRL (未完成)

4.5 Constrained Optimization Algorithms (未完成)

4.5.1 Proxy-Lagrangian Optimization

本节选自论文Two-Player Games for Efficient Non-Convex Constrained Optimization[?] 以及Training Well-Generalizing Classifiers for Fairness Metrics and Other Data-Dependent Constraints[?]

4.6 Summary of Gradient Based Optimizers

1. **算法孰优孰劣尚无定论** 刚入门优先考虑两种方式：
 - SGD + Nesterov Momentum
 - Adam
2. **选择熟练的算法** 这样可以更熟练地利用经验进行调参。
3. **前期选择较大的 batch-size，后期选择较小的 batch-size**
 - 前期选择较大的 batch-size：为了梯度稳定，少走弯路
 - 后期选择较小的 batch-size：为了利用梯度的震荡性，更有可能跳出鞍点
4. **充分了解数据** 如果数据是非常稀疏的，那么优先考虑自适应学习率的算法如：
 - AdaGrad
 - AdaDelta
 - RMSprop
 - Adam
5. **根据需求来选择**
 - 模型设计实验阶段：因为需要快速验证新模型的效果，所以可先用 Adam 算法进行快速实验
 - 模型上线或待发布阶段：因为需要追求极致的效果，所以可用精调的 SGD 进行模型的极致优化
6. **先用小数据集进行实验** 有论文研究指出，优化算法的收敛速度和数据集的大小关系不大。因此可以先用一个具有代表性的小小数据集进行实验，测试一下最好的优化算法，并通过参数搜索寻找最优的优化超参数。
7. **考虑不同算法的组合** 先用 Adam 进行快速下降，而后切换到 SGD 进行充分的调优（有 paper 说明[?]）。
8. **数据集一定要充分打散** 这样在使用自适应算法时候，可以避免学习过度，学习不足，使得下降方向出现偏差的问题。
9. **训练过程中需要持续监控训练集和验证集** 监控目标函数值，RMSE，AUC 等评价准则。其中：
 - 监控训练集：为了保障模型进行充分的训练，即下降方向正确，学习率足够高
 - 监控验证集：为了避免模型训练过拟合
10. **制定一个合适的学习率衰减策略** 可以使用定期衰减策略，比如：
 - 每过 N 个 epoch 就衰减一次
 - 利用精度或 AUC 等性能指标监控，当验证集上的指标不变或者下跌时，就降低学习率
11. **基于动量的优化算法不适用于梯度不稳定的情况，因此此时 RMSprop 为最佳算法** 譬如在 GAN、WGAN 等各种生成对抗网络模型的训练中，梯度十分不稳定，因此不要使用 Momentum, Nesterov 和 Adam 这样基于动量的优化算法，此时 RMSprop 拥有最佳效果

4.7 References

- What's up with Deep Learning optimizers since Adam?
- CSE599s: Online Learning, Spring 2014.

5 Evaluation Metrics

- 评价模型好坏并没有统一的准则
- 评价准则一般针对的是测试数据 (out-samples)，而不是训练数据 (in-samples)
- 具体应用场景中，牵扯多方利益：
 - 用户
 - 供应商/内容生产方
 - 产品运营 (平台)
- 总体而言，模型的评价准则包括三个层次：
 - 算法层面
 - 企业层面
 - 用户层面

5.1 Regression Task

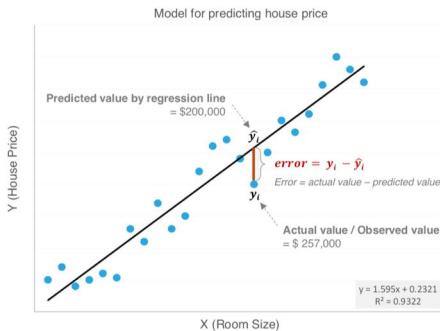


图 31: Error of regression task

5.1.1 Mean Absolute Error (MAE)

$$\text{MAE}(f; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^N |y^{(i)} - \hat{y}^{(i)}|$$

5.1.2 Mean Squared Error (MSE)

$$\text{MSE}(f; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^N (y^{(i)} - \hat{y}^{(i)})^2$$

5.1.3 Root Mean Squared Error (RMSE)

$$\text{RMSE}(f; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^N \sqrt{(y^{(i)} - \hat{y}^{(i)})^2}$$

5.1.4 Mean Absolute Percentage Error (MAPE)

$$\text{MAPE}(f; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^N \frac{|y^{(i)} - \hat{y}^{(i)}|}{|y^{(i)}|}$$

5.1.5 Coefficient of determination (R^2)

$$\begin{aligned} R^2 &= \frac{\sum_{i=1}^N (y^{(i)} - \bar{y})^2}{\sum_{i=1}^N (y^{(i)} - \bar{y})^2 + \sum_{i=1}^N (y^{(i)} - \hat{y}^{(i)})^2} \\ &= 1 - \frac{\sum_{i=1}^N (y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^N (y^{(i)} - \bar{y})^2 + \sum_{i=1}^N (y^{(i)} - \hat{y}^{(i)})^2} \end{aligned}$$

5.2 Classification Task

5.2.1 Log Loss

$$-\sum_{i=1}^N \left(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)}) \right)$$

5.2.2 Confusion Matrix

		Label	
		1	0
Prediction	1	True Positive (TP)	False Positive (FP)
	0	False Negative (FN)	True Negative (TN)

图 32: Confusion matrix for binary classification

- TP: 预测为正类 (Positive), 结果预测对了 (True), 即真实也为正类
- FP: 预测为正类 (Positive), 结果预测错了 (False), 即真实为负类
- FN: 预测为负类 (Negative), 结果预测错了 (False), 即真实为正类
- TN: 预测为负类 (Negative), 结果预测对了 (True), 即真实也为负类

5.2.3 Accuracy

$$\frac{TP + TN}{\#}$$

5.2.4 Precision

所有预测为正的样本中，有多少真实也是正样本

$$\frac{TP}{TP + FP}$$

5.2.5 Recall (Sensitivity)

所有真实为正的样本中，有多少预测也是正样本

$$\frac{TP}{TP + FN}$$

5.2.6 Specificity

所有真实为负的样本中，有多少预测也是负样本

$$\frac{TN}{TN + FP}$$

5.2.7 F1-score

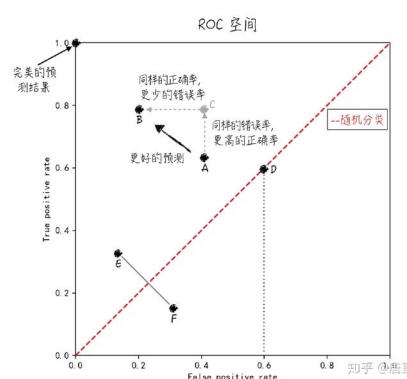
$$\frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

问题：何时使用何种评价准则？

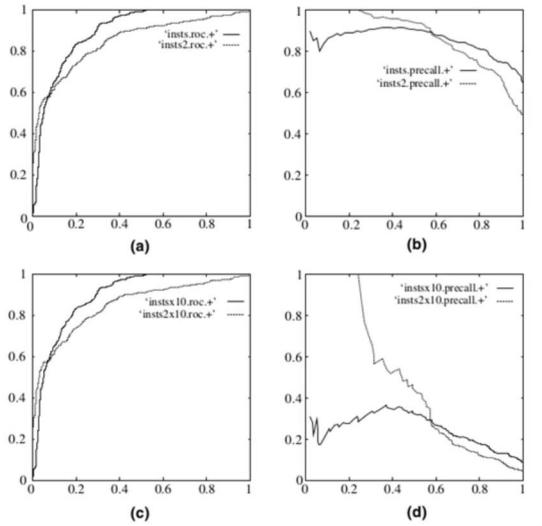
- Accuracy
 - 正负样本数量相对均衡时（猫狗识别样本包含 55% 的猫和 45% 的狗）
 - 正负样本意义权重相当（猫狗分类正样本猫和副样本狗，没有谁更被看中；而癌症的分类的就更看重正样本，不推荐 Accuracy）
- Precision, Recall 强调正样本
- Specificity 强调负样本
- F1-score 均衡 Precision 和 Recall

5.2.8 AUC & ROC

- ROC: 是一种显示分类模型在所有分类阈值（如 Logistic Regression 的阈值，不只是 0.5）下效果的曲线。该曲线的横坐标和纵坐标是两个内容：
 - TPR(True Positive Rate): $TPR = \frac{TP}{TP+FN}$, 即召回率，值越大，在正样本上的准确率越高
 - FPR(False Positive Rate): $FPR = \frac{FP}{FP+TN}$, 值越小，在负样本上的错误率越小，正确率越大
- TPR 越大越好，同时 FPR 越小越好，那么设置横轴为 FPR，纵轴为 TPR，那么曲线越靠近左上角点越好。
- AUC: 即 ROC 曲线下方的面积



问题：既然已经存在很多准则，为何还要使用 ROC 和 AUC？ 因为 ROC 曲线有个很好的特性：当测试集中的正负样本的分布变换的时候，ROC 曲线能够保持不变。在实际的数据集中经常会出现样本类不平衡，即正负样本比例差距较大，而且测试数据中的正负样本也可能随着时间变化。下图是 ROC 曲线和 Precision-Recall 曲线的对比：在图中：



- (a) 和 (c) 为 Roc 曲线
- (b) 和 (d) 为 Precision-Recall 曲线
- (a) 和 (b) 展示的是分类器在原始测试集（正负样本分布平衡）的结果
- (c) 和 (d) 是将测试集中负样本的数量增加到原来的 10 倍后，分类器的结果

可以明显的看出，ROC 曲线基本保持原貌，而 Precision-Recall 曲线变化较大。

AUC 的计算

5.3 Generative Adversarial Task (未完成)

参考自论文An empirical study on evaluation metrics of generative adversarial networks[?]

5.3.1 Inception Score (IS) (未完成)

Definition 5.1. Inception Score(IS) 的公式定义如下：

$$\text{IS}(G) = \exp \left(\mathbb{E}_{\mathbf{x} \sim \mathbb{P}_g} [D_{\text{KL}}(p_{\mathcal{M}}(y|\mathbf{x}) \| p_{\mathcal{M}}(y))] \right)$$

其中：

- $p_{\mathcal{M}}(y|\mathbf{x}) \in \mathbb{P}^K$: 条件标签概率分布 (conditional class distribution)。即将样本 \mathbf{x} 输入某分类判别模型 \mathcal{M} (e.g. softmax model) 得到的标签概率分布
- $p_{\mathcal{M}}(y)$: 边缘标签概率分布 (marginal class distribution)。

$$p_{\mathcal{M}}(y) = \int_{\mathbf{x}} p(y|\mathbf{x}) d\mathbb{P}_g(\mathbf{x})$$

Reference

- A Note on the Inception Score[?]
- CSDN: 全面解析 Inception Score 原理及其局限性

5.3.2 Fréchet Inception Distance (FID)

Definition 5.2. Fréchet Inception Distance(FID) 的公式定义如下：

Reference

- GANs trained by a two time-scale update rule converge to a local Nash equilibrium[?]

5.3.3 Mode Score (MS)

5.3.4 Kernel Maximum Mean Discrepancy (Kernel MMD)

5.3.5 Wasserstein Distance (WD)

6 Model, Feature & Hyper-parameters Selections

6.1 Model Selection

6.1.1 Overfitting & Underfitting

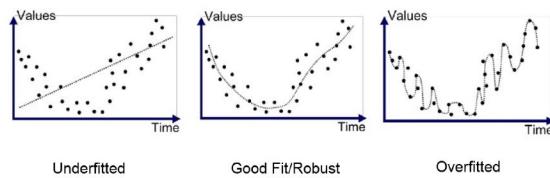


图 33: Underfitting and overfitting

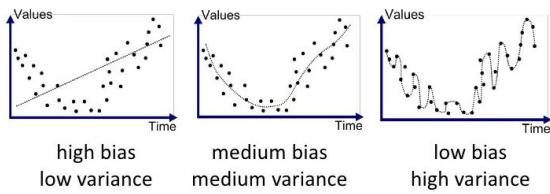


图 34: Underfitting and overfitting correspond to bias and variance

实际中遇到的例子：销量预测，单品的时间序列预测往往不准，因为 high-variance；按类别的销量预测结果会好，因为 low-variance.

6.1.2 Bias-Variance Decomposition

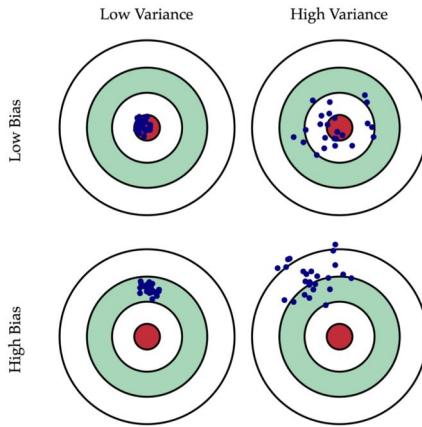


图 35: Bias vs Variance

Introduction Bias 和 Variance 是理解 underfitting 和 overfitting 的一种方式。模型在训练样本上的误差可以分解为如下几个部分：

- Bias: The part of the models can't fit the data
- Variance: The part of the models could fit the data, but doesn't because it's hard to fit
- Noise: Randomly error can't be controlled

Proof 假定一个真正的预测函数在我们预测前是已经存在的（上帝创造出来放在那里，等着我们去发现的函数），也就是我们希望可以训练出的最完美的函数，记为：

$$y = f(\mathbf{x})$$

我们希望通过拟合现有训练数据集学到这个上帝函数 $f(\mathbf{x})$ ，训练数据集记作：

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^N$$

注意其中 $t^{(i)}$ 是夹杂了随机噪声的标签：

$$t^{(i)} = y^{(i)} + \epsilon, \quad \mathbb{E}(\epsilon) = 0 \quad \forall i \in \{1, \dots, n\}$$

那么根据给定的训练数据集，我们希望训练出一个模型去近似拟合真实的上帝函数 $f(\mathbf{x})$ （梦想还是要有的，万一实现了呢），记根据训练数据 \mathcal{D} 训练出的模型函数为：

$$\phi = \phi(\boldsymbol{\theta}; \mathcal{D})$$

我们使用 MSE 期望表示模型在训练样本上的误差，即：

$$\begin{aligned} \text{error} &= \mathbb{E} \left\{ \frac{1}{N} \sum_{i=1}^N (t^{(i)} - \phi^{(i)})^2 \right\} \quad (\text{注意从训练数据中观测到的标签是 } t^{(i)}, \text{ 而不是 } y^{(i)}) \\ &= \frac{1}{N} \sum_{i=1}^N \mathbb{E} \{ (t^{(i)} - \phi^{(i)})^2 \} \end{aligned}$$

问题：为什么是 MSE 的期望而不是 MSE ? 因为譬如模型为神经网络时，改变训练数据的顺序，都可能导致模型的预测函数变化，因此用于之后测试阶段时，对同样的测试样本会有不同的输出，所以一次的预测结果没有足够的说服力，只能多测几次，求期望。

那么对于任何一个训练样本 $(x^{(i)}, t^{(i)}) \forall i \in \{1, \dots, n\}$ ，为了符号表示的便捷性，不写索引记为 (x, t) ，其期望平方误差可以做如下拆解：

$$\begin{aligned}
 \mathbb{E}\{(t - \phi)^2\} &= \mathbb{E}\{(t - y + y - \phi)^2\} \quad (\text{trick1: 添加上帝函数值 } y) \\
 &= \mathbb{E}\{(t - y)^2 + (y - \phi)^2 + 2(t - y)(y - \phi)\} \\
 &= \underbrace{\mathbb{E}\{(t - y)^2\}}_{\epsilon} + \mathbb{E}\{(y - \phi)^2\} + 2\mathbb{E}\{(t - y)(y - \phi)\} \\
 &= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y - \phi)^2\} + 2\mathbb{E}\{ty - y^2 - t\phi + y\phi\} \\
 &= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y - \phi)^2\} + 2(\mathbb{E}\{ty\} - \mathbb{E}\{y^2\} - \mathbb{E}\{t\phi\} + \mathbb{E}\{y\phi\}) \\
 \therefore \mathbb{E}\{ty\} &= y\mathbb{E}\{t\} \quad (\because y \text{ 为上帝 label, 是已知的常数}) \\
 &= y\mathbb{E}\{(y + \epsilon)\} \quad (\because t = y + \epsilon) \\
 &= y\mathbb{E}\{y\} + y\mathbb{E}\{\epsilon\} = y^2 \quad (\because \text{常数的期望为常数}) \\
 \therefore \mathbb{E}\{y^2\} &= y\mathbb{E}\{y\} \quad (\text{先提一个常数 } y, \text{ 之后再提另一个}) \\
 &= y^2 \\
 \therefore \mathbb{E}\{t\phi\} &= \mathbb{E}\{(y + \epsilon)\phi\} \quad (\because t = y + \epsilon) \\
 &= \mathbb{E}\{y\phi\} + \mathbb{E}\{\epsilon\phi\} \\
 &= \mathbb{E}\{y\phi\} \quad (\because \epsilon \text{ 和 } \phi \text{ 相互独立}, \therefore \mathbb{E}\{\epsilon\phi\} = 0) \\
 \therefore \mathbb{E}\{(t - \phi)^2\} &= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y - \phi)^2\} \quad (\text{trick2: 添加预测函数 } \phi \text{ 的期望 } \mathbb{E}\{\phi\}) \\
 &= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y - \mathbb{E}\{\phi\} + \mathbb{E}\{\phi\} - \phi)^2\} \\
 &= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y - \mathbb{E}\{\phi\})^2 + (\mathbb{E}\{\phi\} - \phi)^2 + 2(y - \mathbb{E}\{\phi\})(\mathbb{E}\{\phi\} - \phi)\} \\
 &= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y - \mathbb{E}\{\phi\})^2\} + \mathbb{E}\{(\mathbb{E}\{\phi\} - \phi)^2\} + 2\mathbb{E}\{(y - \mathbb{E}\{\phi\})(\mathbb{E}\{\phi\} - \phi)\} \\
 &= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y - \mathbb{E}\{\phi\})^2\} + \mathbb{E}\{(\mathbb{E}\{\phi\} - \phi)^2\} \\
 &\quad + 2\mathbb{E}\{y\mathbb{E}\{\phi\} - (\mathbb{E}\{\phi\})^2 - y\phi + \mathbb{E}\{\phi\}\phi\} \\
 &= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y - \mathbb{E}\{\phi\})^2\} + \mathbb{E}\{(\mathbb{E}\{\phi\} - \phi)^2\} \\
 &\quad + 2\underbrace{(\mathbb{E}\{y\mathbb{E}\{\phi\}\} - \mathbb{E}\{(\mathbb{E}\{\phi\})^2\} - \mathbb{E}\{y\phi\} + \mathbb{E}\{\mathbb{E}\{\phi\}\phi\})}_{\text{该项为 } 0} \\
 \therefore \mathbb{E}\{y\mathbb{E}\{\phi\}\} &= y\mathbb{E}\{\mathbb{E}\{\phi\}\} \quad (\because y \text{ 为常数, 可提出}) \\
 &= y\mathbb{E}\{\phi\} \quad (\because \text{期望 } \mathbb{E}\{\phi\} \text{ 是常数} \therefore \mathbb{E}\{\mathbb{E}\{\phi\}\} = \mathbb{E}\{\phi\}) \\
 \mathbb{E}\{y\phi\} &= y\mathbb{E}\{\phi\} \\
 \mathbb{E}\{(\mathbb{E}\{\phi\})^2\} &= \mathbb{E}\{\phi\}\mathbb{E}\{\mathbb{E}\{\phi\}\} \quad (\because \text{期望 } \mathbb{E}\{\phi\} \text{ 是常数}) \\
 &= (\mathbb{E}\{\phi\})^2 \\
 \mathbb{E}\{\mathbb{E}\{\phi\}\phi\} &= \mathbb{E}\{\phi\}\mathbb{E}\{\phi\} = (\mathbb{E}\{\phi\})^2 \\
 \therefore \mathbb{E}\{(t - \phi)^2\} &= \underbrace{\mathbb{E}\{\epsilon^2\}}_{\text{noise}} + \underbrace{\mathbb{E}\{(y - \mathbb{E}\{\phi\})^2\}}_{\text{Bias}} + \underbrace{\mathbb{E}\{(\mathbb{E}\{\phi\} - \phi)^2\}}_{\text{Variance}}
 \end{aligned}$$

所以最终模型在训练数据集上的误差被拆分成了偏差，方差和噪声三项内容。

Bias-variance decomposition note

- 第一次拆解：引入上帝 label: y
- 第二次拆解：引入预测函数的期望 $\mathbb{E}\{\phi\}$
- Bias-Variance Decomposition 是在训练集上的错误分解，而不是测试集上的错误分解

6.2 Feature Selection

特征选择依据思路不同，分为以下三类：

- **Filter:** 自变量和目标变量之间的关系
- **Wrapper:** 通过目标函数（譬如：mse，logloss）或评价准则（譬如：AUC）决定是否加入一个变量
- **Embedded:** 学习器自身自动选择特征

6.2.1 Filter methods

6.2.2 Wrapper methods

Note of wrapper methods 实际操作中具体的流程是 (group \Rightarrow field) 层次模式的：

1. group 级别测试（如：所有的天气 fields（最高温，最低温，PM2.5，天气类型）为一个天气 group）
 - 选定一个基础的 group G_{base} ，测试模型在 G_{base} 上的模型效果
 - 添加一个候选 group $G_{candidate}$ ，测试模型在 $\{G_{base}, G_{candidate}\}$ 上的模型效果
2. field 级别测试
 - 对同一确定候选 group $G_{candidate}$ 内所有的单个 field $f_{candidate, i}$ 进行测试
 - 对同一确定候选 group $G_{candidate}$ 内所有的组合 fields $\{f_{candidate, i_1}, \dots, f_{candidate, i_m}\}$ 进行测试
 - 对不同候选 group 组合 $\{G_{candidate_1}, \dots, G_{candidate_n}\}$ 内的所有组合 fields $\{f_{candidate_1, i_1}, \dots, f_{candidate_n, i_m}\}$ 进行测试

6.2.3 Embedded methods

6.3 Hyper-parameters Selection

6.3.1 Model validation strategies

- Hold-out validation
- K-fold cross validation
- Leave-one-out cross validation

Part II

Machine Learning Model and Theory

7 Logistic Regression

7.1 Logistic Regression

7.1.1 Model prediction

Logistic Regression 是一个用于二分类问题的线性模型，预测判别函数可表示为：

$$\phi(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(-\sum_{j=1}^d w_j x_j)}$$

其中判别函数包括两部分：

1. 内函数 (linear function): $z = \mathbf{w}^T \mathbf{x} = \sum_{j=1}^d w_j x_j$

2. 外函数 (sigmoid function): $\phi(z) = \frac{1}{1 + \exp(-z)}$

3. 合并表示: $\phi(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(-\sum_{j=1}^d w_j x_j)}$

判别类别条件为：

$$h(\mathbf{x}) = \begin{cases} 0, & \text{if } \phi(\mathbf{x}; \mathbf{w}) \leq \alpha \\ 1, & \text{if } \phi(\mathbf{x}; \mathbf{w}) > \alpha \end{cases}$$

其中 α 为判别罚值，一般有 $\alpha = 0.5$ 。

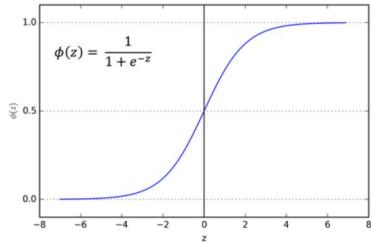


图 36: Sigmoid function

待学习参数 Logistic Regression 的待学习参数为：

$$\mathbf{w} \in \mathbb{R}^d$$

其中 $w_j, j \in \{1, \dots, d\}$ 表示原始数据第 j 个维度上的权重。

7.1.2 Model training

Dataset 训练数据的表达如下：

- 训练数据集: $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$
- 样本标签: $y^{(i)} \in \{0, 1\}$

Modeling details Logistic Regression 的判别函数包括两部分：

1. 内函数 (linear function): $z = \mathbf{w}^T \mathbf{x} = \sum_{j=1}^d w_j x_j$

2. 外函数 (sigmoid function): $\phi(z) = \frac{1}{1 + \exp(-z)}$

3. 合并表示: $\phi(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(-\sum_{j=1}^d w_j x_j)}$

Loss function 以下从两个角度推导 Logistic Regression 的目标函数 (求梯度之前的所有建模步骤) :

- Maximum Likelihood Estimation
- Kullback–Leibler-Divergence

Maximum Likelihood Estimation 为了数学公式表达的简便，我们做如下符号记录：

- $p(\cdot)$: 事件 · 发生的概率
- 样本事件 $\mathbf{x}^{(i)}$ 发生 ($y^{(i)} = 1$) 的估计概率：

$$q^{(i)} = p(y^{(i)} = 1 | \mathbf{x}^{(i)})$$

- 样本事件 $\mathbf{x}^{(i)}$ 不发生 ($y^{(i)} = 0$) 的估计概率：

$$\begin{aligned} 1 - q^{(i)} &= p(y^{(i)} = 0 | \mathbf{x}^{(i)}) \\ &= 1 - p(y^{(i)} = 1 | \mathbf{x}^{(i)}) \end{aligned}$$

整合二者，利用 0/1 表示二分类标签的便利性，可得训练样本 (带标签的样本事件) $(\mathbf{x}^{(i)}, y^{(i)})$ 出现的估计概率为：

$$(q^{(i)})^{y^{(i)}} (1 - q^{(i)})^{1-y^{(i)}}$$

假设所有的训练样本都是 **独立同分布** (independently and identically distributed) 的，那么所有训练样本都出现的概率为：

$$\prod_{i=1}^N ((q^{(i)})^{y^{(i)}} (1 - q^{(i)})^{1-y^{(i)}})$$

那么模型的优化目标就是最大化所有样本都出现的概率，即：

$$\begin{aligned} \max & \prod_{i=1}^N ((q^{(i)})^{y^{(i)}} (1 - q^{(i)})^{1-y^{(i)}}) \\ \implies \max & \underbrace{\log \left(\prod_{i=1}^N ((q^{(i)})^{y^{(i)}} (1 - q^{(i)})^{1-y^{(i)}}) \right)}_{\text{对目标函数取对数}} \\ \implies \max & \sum_{i=1}^N \log ((q^{(i)})^{y^{(i)}} (1 - q^{(i)})^{1-y^{(i)}}) \end{aligned}$$

$$\begin{aligned}
&\Rightarrow \max \sum_{i=1}^N (\log(q^{(i)})^{y^{(i)}} + \log(1 - q^{(i)})^{1-y^{(i)}}) \\
&\Rightarrow \max \sum_{i=1}^N (y^{(i)} \log(q^{(i)}) + (1 - y^{(i)}) \log(1 - q^{(i)})) \\
&\Rightarrow \min -\frac{1}{N} \sum_{i=1}^N (y^{(i)} \log(q^{(i)}) + (1 - y^{(i)}) \log(1 - q^{(i)})) \\
&\Rightarrow \min -\frac{1}{N} \sum_{i=1}^N (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})) \quad (\because \hat{y}^{(i)} = q^{(i)})
\end{aligned}$$

因此最终的训练损失函数为：

$$\min -\frac{1}{N} \sum_{i=1}^N (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$

Kullback–Leibler-Divergence 为了数学符号表示的便捷性，我们做如下符号表示：

- p : 样本类别的真实概率分布
- q : 样本类别的估计概率分布

根据 KL-Divergence 进行目标函数的推导，实际上是希望最小化所有训练样本类别的真实概率分布和估计概率分布的差异，即：

$$\begin{aligned}
&\min \frac{1}{N} \sum_{i=1}^N D_{\text{KL}}(p^{(i)} \parallel q^{(i)}) \\
&\Rightarrow \min \frac{1}{N} \sum_{i=1}^N (p^{(i)} \cdot \log(\frac{p^{(i)}}{q^{(i)}})) \\
&\Rightarrow \min \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log(\frac{p_k^{(i)}}{q_k^{(i)}}) \\
&\Rightarrow \min \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} (\log(p_k^{(i)}) - \log(q_k^{(i)})) \\
&\Rightarrow \min \underbrace{\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log(p_k^{(i)})}_{\text{常数，可忽略}} - \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log(q_k^{(i)}) \\
&\Rightarrow \min -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log(q_k^{(i)}) \\
&\Rightarrow \min -\frac{1}{N} \sum_{i=1}^N (y^{(i)} \log(q^{(i)}) + (1 - y^{(i)}) \log(1 - q^{(i)})) \\
&\Rightarrow \min -\frac{1}{N} \sum_{i=1}^N (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})) \quad (\because \hat{y}^{(i)} = q^{(i)})
\end{aligned}$$

因此最终的训练损失函数为：

$$\min -\frac{1}{N} \sum_{i=1}^N (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$

Optimization 引入 l2 正则化项，Logistic Regression 模型的目标函数为：

$$J(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

为了之后计算的便捷性，先对 $\phi(z)$ 求梯度：

$$\begin{aligned} \frac{\partial \phi(z)}{\partial z} &= -(1 + e^{-z})^{-2} e^{-z} (-1) \\ &= \frac{e^{-z}}{(1 + e^{-z})^2} \\ &= \frac{e^{-z}}{1 + e^{-z}} \cdot \frac{1}{1 + e^{-z}} \\ &= (1 - \phi(z))\phi(z) \end{aligned}$$

对第 j 个维度的参数求梯度：

$$\begin{aligned} -\frac{\partial J(\mathbf{w}; \mathbf{x}^{(i)}; y^{(i)})}{\partial w_j} &= \frac{\partial}{\partial w_j} \left(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right) - \frac{\partial}{\partial w_j} \left(\frac{\lambda}{2} \mathbf{w}^T \mathbf{w} \right) \\ &= \frac{y^{(i)}}{\hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial w_j} + \frac{1 - y^{(i)}}{1 - \hat{y}^{(i)}} (-1) \frac{\partial \hat{y}^{(i)}}{\partial w_j} - \lambda w_j \\ &= \frac{\hat{y}^{(i)}(y^{(i)} - 1) + y^{(i)}(1 - \hat{y}^{(i)})}{\hat{y}^{(i)}(1 - \hat{y}^{(i)})} \frac{\partial \hat{y}^{(i)}}{\partial w_j} - \lambda w_j \end{aligned}$$

根据之前对 $\phi(z)$ 所求梯度，运用梯度链式法则有：

$$\frac{\partial \hat{y}^{(i)}}{\partial w_j} = \frac{\partial \phi(\mathbf{x}^{(i)}; \mathbf{w})}{\partial w_j} = \frac{\partial \phi(z^{(i)})}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial w_j}$$

因此代入上式中可继续化简得：

$$\begin{aligned} -\frac{\partial J(\mathbf{w}; \mathbf{x}^{(i)}; y^{(i)})}{\partial w_j} &= \frac{\hat{y}^{(i)}(y^{(i)} - 1) + y^{(i)}(1 - \hat{y}^{(i)})}{\hat{y}^{(i)}(1 - \hat{y}^{(i)})} \frac{\partial \hat{y}^{(i)}}{\partial w_j} - \lambda w_j \\ &= \frac{\hat{y}^{(i)}(y^{(i)} - 1) + y^{(i)}(1 - \hat{y}^{(i)})}{\hat{y}^{(i)}(1 - \hat{y}^{(i)})} \frac{\partial \phi(z)}{\partial z} \frac{\partial z}{\partial w_j} - \lambda w_j \\ &= \frac{\hat{y}^{(i)}(y^{(i)} - 1) + y^{(i)}(1 - \hat{y}^{(i)})}{\hat{y}^{(i)}(1 - \hat{y}^{(i)})} (1 - \phi(z^{(i)}))\phi(z^{(i)})x_j^{(i)} - \lambda w_j \\ &= \frac{\hat{y}^{(i)}y^{(i)} - \hat{y}^{(i)} + y^{(i)} - \hat{y}^{(i)}y^{(i)}}{\hat{y}^{(i)}(1 - \hat{y}^{(i)})} (1 - \hat{y}^{(i)})\hat{y}^{(i)}x_j^{(i)} - \lambda w_j \quad (\because \phi(z^{(i)}) = \hat{y}^{(i)}) \\ &= (y^{(i)} - \hat{y}^{(i)})x_j^{(i)} - \lambda w_j \end{aligned}$$

因此使用 Mini-Batch-SGD 优化算法时迭代更新公式为：

$$\begin{aligned} w_j &\leftarrow w_j - \eta \left(\frac{1}{m} \sum_{i=1}^m \frac{\partial J(\mathbf{w}; \mathbf{x}^{(i)}; y^{(i)})}{\partial w_j} \right) \\ w_j &\leftarrow w_j + \eta \left(\frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})x_j^{(i)} - \lambda w_j \right) \end{aligned}$$

7.1.3 Model implement

```

# -*- coding: utf-8 -*-
"""
Created on 2018/10/31 05:03
author: Tong Jia
email: cecilio.jia@gmail.com
software: PyCharm
Description:
    An Implement of Logistic Regression Based on TensorFlow.

"""

import tensorflow as tf
from tensorflow.contrib.layers import l2_regularizer

def input_fn(
    filenames,
    delimiter,
    batch_size,
    epochs,
    identifier=":",
    perform_shuffle=True,
    dtype_indices=tf.int32,
    dtype_values=tf.float32,
    name_feature_indices="feat_inds",
    name_feature_values="feat_vals",
    num_parallel_calls=10,
    buffer_size_prefetch=500000,
    buffer_size_shuffle=2048):
    """
Description
-----
The input function for loading training dataset when using tensorflow high level API Estimator. The columns
are
delimiter(e.g. tab) separated with the following schema:
<label> <index 1>:<value 1> ... <index j>:<value j> ... <index d>:<value d> (d is the field size)
where:
    delimiter is "\t" in normal txt file, "," in normal csv file;
    identifier always equals to ":".

Parameters
-----
:param filenames: A list containing one or more filenames.
:param delimiter: A str(e.g. "\t") to separate different <feature index j>:<feature value j> pairs in
dataset files.
:param batch_size: An integer scalar, representing the number of consecutive elements of this dataset to
combine in a single batch.
:param epochs: An integer scalar, representing the number of times the dataset should be repeated.
:param identifier: A str(e.g. defaults to ":") to seprate feature index and feature value in one specific
pair.
:param perform_shuffle: A bool variable(defaults to True) to instruct whether randomly shuffles the
elements of this dataset.
:param dtype_indices: The numerical type of indices. Note it must be equal to dtype_ind in model_fn.
:param dtype_values: The numerical type of values. Note it must be equal to dtype_val in model_fn.
:param name_feature_indices: A str, representing the name of feature indices in return.

```

```

:param name_feature_values: A str, representing the name of feature values in return.
:param num_parallel_calls: A integer scalar, representing the number elements to process in parallel.
:param buffer_size_prefetch: A integer scalar, representing the maximum number of elements that will be
                                buffered when prefetching.
:param buffer_size_shuffle: A integer scalar, representing the number of elements from this dataset from
                                which the new dataset will sample.

>Returns
-----
:return: A dict of two Tensors, representing feature indices and feature values in a single batch.
{
    <name_feature_indices>: feature indices Tensor in shape of (batch_size, field_size),
    <name_feature_values>: feature values Tensor in shape of (batch_size, field_size)
}
:return: A Tensor in shape of (batch_size, ), representing labels in a single batch.
"""

def map_func(line):
    columns = tf.string_split(source=[line], delimiter=delimiter, skip_empty=False)
    label = tf.string_to_number(string_tensor=columns.values[0], out_type=dtype_values)
    splits = tf.string_split(source=columns.values[1: ], delimiter=identifier, skip_empty=False)
    features = tf.reshape(tensor=splits.values, shape=splits.dense_shape) # A tensor in shape of
                                                                (field_size, 2), the first column are feature
                                                                indices, and the second column are feature values
    feat_inds_str, feat_vals_str = tf.split(value=features, num_or_size_splits=2, axis=1) # Two tensors in
                                                                shape of (field_size, 1)
    feat_inds = tf.string_to_number(string_tensor=feat_inds_str, out_type=dtype_indices) # A tensor in shape
                                                                of (field_size, 1)
    feat_vals = tf.string_to_number(string_tensor=feat_vals_str, out_type=dtype_values) # A tensor in shape
                                                                of (field_size, 1)
    feat_inds = tf.reshape(tensor=feat_inds, shape=(-1, )) # A tensor in shape of (field_size, )
    feat_vals = tf.reshape(tensor=feat_vals, shape=(-1, )) # A tensor in shape of (field_size, )
    return {name_feature_indices: feat_inds, name_feature_values: feat_vals}, label

dataset = tf.data.TextLineDataset(filenames=filenames). \
    map(map_func=map_func, num_parallel_calls=num_parallel_calls). \
    prefetch(buffer_size=buffer_size_prefetch)
if perform_shuffle ==True:
    dataset = dataset.shuffle(buffer_size=buffer_size_shuffle)
dataset = dataset. \
    repeat(count=epoches). \
    batch(batch_size=batch_size, drop_remainder=False)
iterator = dataset.make_one_shot_iterator()
features, labels = iterator.get_next()
return features, labels

def model_fn(features, labels, mode, params):
    """Model function of Logistic Regression model for sparse predictive analytics."""
    # -----hyper-parameters-----
    feature_size = params["feature_size"]
    use_global_bias = params["use_global_bias"]
    l2_reg = params["l2_reg"]

```

```

optimizer =params["optimizer"]
learning_rate =params["learning_rate"]
name_feature_indices =params["name_feature_indices"]
name_feature_values =params["name_feature_values"]
dtype_indices =params["dtype_indices"]
dtype_values =params["dtype_values"]

# -----features-----
# with tf.variable_scope(name_or_scope="features"):
feat_inds =features[name_feature_indices] # A tensor in shape of (None, field_size)
feat_vals =features[name_feature_values] # A tensor in shape of (None, field_size)

# -----Build f(x)-----
with tf.variable_scope(name_or_scope="parameters"):
    # -----parameters-----
    w0 =tf.get_variable(
        name="w0",
        shape=[1, ],
        dtype=dtype_values,
        initializer=tf.zeros_initializer(dtype=dtype_values)
    ) # The variable of global bias
    W =tf.get_variable(
        name="W",
        shape=[feature_size, ],
        dtype=dtype_values,
        initializer=tf.truncated_normal_initializer(mean=0.0, stddev=0.01, dtype=dtype_values),
        regularizer=l2_regularizer(scale=l2_reg)
    ) # The variables of weights (first order)

with tf.variable_scope(name_or_scope="embedding-lookup"):
    # embedding lookup operations for weights
    w =tf.nn.embedding_lookup(params=W, ids=feat_inds) # A tensor in shape of (None, field_size)

with tf.variable_scope(name_or_scope="linear-part"):
    y_lr =tf.reduce_sum(
        input_tensor=tf.multiply(x=w, y=feat_vals), # A tensor in shape of (None, field_size),
        axis=1,
        keepdims=False
    ) # A tensor in shape of (None, )

with tf.variable_scope(name_or_scope="output"):
    if use_global_bias ==True:
        logits =w0 +y_lr
    else:
        logits =y_lr
    # A tensor in shape of (None, )

predictions =[

    # Generate predictions (for PREDICT and EVAL mode)
    "probability": tf.sigmoid(x=logits),
    "class": tf.cast(x=tf.round(x=tf.sigmoid(x=logits)), dtype=dtype_indices)
]

```

```

    export_outputs ={
        tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
            tf.estimator.export.PredictOutput(predictions)
    }

    # -----Provide an estimator spec for `ModeKeys.PREDICTION` mode-----
    if mode ==tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions, export_outputs=export_outputs)

    # -----Build loss function (for both TRAIN and EVAL modes)-----
    with tf.variable_scope(name_or_scope="loss"):

        loss =tf.reduce_mean(
            input_tensor=tf.nn.sigmoid_cross_entropy_with_logits(labels=labels, logits=logits), # A tensor in
            shape of (None, )
            axis=0,
            keepdims=False
        ) # A scalar representing the residual loss
        reg_loss =tf.reduce_sum(
            input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
            axis=0,
            keepdims=False
        ) # A scalar representing the regularization loss
        loss +=reg_loss

    # -----Build optimizer-----
    with tf.variable_scope(name_or_scope="optimization"):

        if (optimizer.lower() == "sgd"):
            opt =tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
        elif (optimizer.lower() == "momentum"):
            opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=False)
        elif (optimizer.lower() == "nesterov"):
            opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=True)
        elif (optimizer.lower() == "adagrad"):
            opt =tf.train.AdagradOptimizer(learning_rate=learning_rate, initial_accumulator_value=0.1)
        elif (optimizer.lower() == "adadelta"):
            opt =tf.train.AdadeltaOptimizer(learning_rate=learning_rate, rho=0.95, epsilon=1e-8)
        elif (optimizer.lower() == "rmsprop"):
            opt =tf.train.RMSPropOptimizer(learning_rate=learning_rate, decay=0.9, epsilon=1e-10)
        elif (optimizer.lower() == "ftrl"):
            opt =tf.train.FtrlOptimizer(learning_rate=learning_rate)
        elif (optimizer.lower() == "adam"):
            opt =tf.train.AdamOptimizer(learning_rate=learning_rate, beta1=0.9, beta2=0.999, epsilon=1e-8)
        else:
            raise ValueError("Argument <optimizer> of model_fn is invalid")

        train_op =opt.minimize(loss=loss, global_step=tf.train.get_global_step())

    # -----Provide an estimator spec for `ModeKeys.TRAIN` mode-----
    if (mode ==tf.estimator.ModeKeys.TRAIN):
        return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

    # -----Build evaluation metrics-----

```

```
with tf.variable_scope(name_or_scope="evaluation"):  
    eval_metric_ops = {  
        "accuracy": tf.metrics.accuracy(labels=labels, predictions=predictions["class"]),  
        "precision": tf.metrics.precision(labels=labels, predictions=predictions["class"]),  
        "recall": tf.metrics.recall(labels=labels, predictions=predictions["class"]),  
        "auc": tf.metrics.auc(labels=labels, predictions=predictions["class"])}  
    }  
  
# -----Provide an estimator spec for `ModeKeys.EVAL` mode-----  
if mode ==tf.estimator.ModeKeys.EVAL:  
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, predictions=predictions,  
                                      eval_metric_ops=eval_metric_ops)
```

7.2 Softmax Regression

7.2.1 Model prediction

Softmax Regression，又称 Multinomial Logistic Regression，是一个用于多分类的线性模型，其本质是将多分类问题转换为多分类类别的离散型概率分布问题。判别函数包括两部分：

- 内函数：每类的 logit 函数：

$$z_k = \sum_{j=1}^d w_{k,j} x_j, \quad k \in \{1, \dots, K\}$$

- 外函数：Softmax 函数，即每类的概率估计（注意指数函数中没有负号）：

$$q_k = \frac{\exp(z_k)}{\sum_{l=1}^K \exp(z_l)}, \quad k \in \{1, \dots, K\}$$

- 合并表示：第 k 类的概率估计：

$$q_k = \frac{\exp(\sum_{j=1}^d w_{k,j} x_j)}{\sum_{l=1}^K \exp(\sum_{j=1}^d w_{l,j} x_j)} = p(y = k | \mathbf{x})$$

其中 $w_{k,j}$ 表示，特征向量 \mathbf{x} 的第 j 维到输出第 k 类之间的参数。计算完每类的概率，概率最大的类别就是

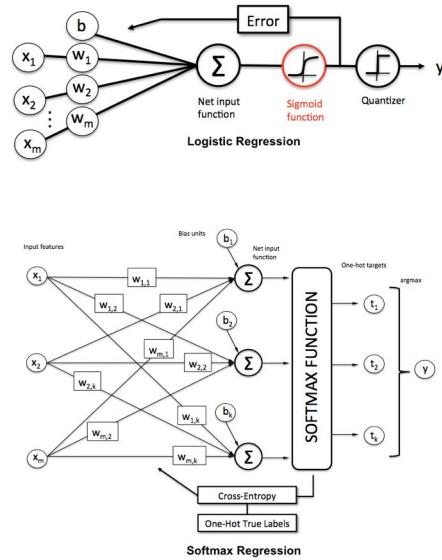


图 37: Softmax regression & logistic regression

最终的预测类别。

待学习参数 Softmax Regression 的待学习参数为矩阵：

$$w \in \mathbb{R}^{K \times d}$$

其中 $w_{k,j}, k \in \{1, \dots, K\}, j \in \{1, \dots, d\}$ 表示链接特征向量 \mathbf{x} 的第 j 个维度和最终分类第 k 类的参数。

7.2.2 Model training

Dataset 训练数据的表达如下：

- 训练数据集： $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$
- 样本标签： $y^{(i)} \in \mathbb{R}^K$

Modeling details Softmax Regression 的判别函数包括两部分：

- 内函数：每类的 logit 函数：

$$z_k = \sum_{j=1}^d w_{k,j} x_j, \quad k \in \{1, \dots, K\}$$

- 外函数：Softmax 函数，即每类的概率估计：

$$q_k = \frac{\exp(z_k)}{\sum_{l=1}^K \exp(z_l)}, \quad k \in \{1, \dots, K\}$$

- 合并表示：第 k 类的概率估计：

$$q_k = \frac{\exp(\sum_{j=1}^d w_{k,j} x_j)}{\sum_{l=1}^K \exp(\sum_{j=1}^d w_{l,j} x_j)}$$

Loss function 根据 KL-Divergence 进行表示可得：

$$\begin{aligned} & \min \quad \frac{1}{N} \sum_{i=1}^N D_{\text{KL}}(\mathbf{p}^{(i)} \parallel \mathbf{q}^{(i)}) \\ \implies & \min \quad \frac{1}{N} \sum_{i=1}^N \left(\mathbf{p}^{(i)} \cdot \log\left(\frac{\mathbf{p}^{(i)}}{\mathbf{q}^{(i)}}\right) \right) \\ \implies & \min \quad \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log\left(\frac{p_k^{(i)}}{q_k^{(i)}}\right) \\ \implies & \min \quad \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} (\log(p_k^{(i)}) - \log(q_k^{(i)})) \\ \implies & \min \quad \underbrace{\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log(p_k^{(i)})}_{\text{熵, 常数}} + \underbrace{\left(-\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log(q_k^{(i)}) \right)}_{\text{交叉熵}} \\ \implies & \min \quad -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log(q_k^{(i)}) \end{aligned}$$

注意：

- 之后推导分类问题的损失函数时，只写交叉熵 (Cross Entropy) 即可，不必再从 KL-Divergence 起步
- Maximum Log Likelihood \Leftrightarrow Minimize KL-Divergence**

Optimization 引入 l2 正则化项，Softmax Regression 模型的目标函数为：

$$\begin{aligned} J(\mathbf{w}) &= -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K p_k^{(i)} \log(q_k^{(i)}) + \frac{\lambda}{2} \sum_{k=1}^K \sum_{j=1}^d w_{k,j}^2 \\ &= -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K \underbrace{I(y^{(i)} = k)}_{0/1 \text{ 独热离散型分布}} \log(q_k^{(i)}) + \frac{\lambda}{2} \sum_{k=1}^K \sum_{j=1}^d w_{k,j}^2 \end{aligned}$$

对联系输入第 j 个维度和输出第 k 类的参数 $w_{k,j}$ 求梯度，可得：

$$\frac{\partial J(\mathbf{w}; \mathbf{x}^{(i)}; y^{(i)})}{\partial w_{k,j}} = -I(y^{(i)} = k) \frac{1}{q_k^{(i)}} \frac{\partial q_k^{(i)}}{\partial w_{k,j}} + \lambda w_{k,j}$$

根据链式求导法则可得：

$$\frac{\partial q_k^{(i)}}{\partial w_{k,j}} = \frac{\partial q_k^{(i)}}{\partial z_k^{(i)}} \frac{\partial z_k^{(i)}}{\partial w_{k,j}}$$

为了简化表示暂时不考虑样本索引 i ，因为：

$$\begin{aligned} \frac{\partial q_k}{\partial z_k} &= \frac{\partial}{\partial z_k} \frac{\exp(z_k)}{\sum_{l=1}^K \exp(z_l)} \quad (\because q_k = \frac{\exp(z_k)}{\sum_{l=1}^K \exp(z_l)}) \\ &= \frac{\exp(z_k) \sum_{l=1}^K \exp(z_l) - \exp(z_k) \exp(z_k)}{\left(\sum_{l=1}^K \exp(z_l) \right)^2} \\ &= \frac{\exp(z_k) \left(\sum_{l=1}^K \exp(z_l) - \exp(z_k) \right)}{\left(\sum_{l=1}^K \exp(z_l) \right)^2} \\ &= \frac{\exp(z_k)}{\sum_{l=1}^K \exp(z_l)} \times \frac{\sum_{l=1}^K \exp(z_l) - \exp(z_k)}{\sum_{l=1}^K \exp(z_l)} \\ &= \frac{\exp(z_k)}{\sum_{l=1}^K \exp(z_l)} \times \left(1 - \frac{\exp(z_k)}{\sum_{l=1}^K \exp(z_l)} \right) \\ &= q_k (1 - q_k) \\ \frac{\partial z_k^{(i)}}{\partial w_{k,j}} &= \frac{\partial}{\partial w_{k,j}} \sum_{j=1}^d w_{k,j} x_j \quad (\because z_k = \sum_{j=1}^d w_{k,j} x_j) \\ &= x_j \end{aligned}$$

所以代入上式可得：

$$\begin{aligned} \frac{\partial q_k^{(i)}}{\partial w_{k,j}} &= \frac{\partial q_k^{(i)}}{\partial z_k^{(i)}} \frac{\partial z_k^{(i)}}{\partial w_{k,j}} \\ &= q_k^{(i)} (1 - q_k^{(i)}) x_j^{(i)} \end{aligned}$$

$$\begin{aligned}
\frac{\partial J(\mathbf{w}; \mathbf{x}^{(i)}; y^{(i)})}{\partial w_{k,j}} &= -I(y^{(i)} = k) \frac{1}{q_k^{(i)}} \frac{\partial q_k^{(i)}}{\partial w_{k,j}} + \lambda w_{k,j} \\
&= -I(y^{(i)} = k) \frac{1}{q_k^{(i)}} q_k^{(i)} (1 - q_k^{(i)}) x_j^{(i)} + \lambda w_{k,j} \\
&= -I(y^{(i)} = k) (1 - q_k^{(i)}) x_j^{(i)} + \lambda w_{k,j} \\
&= -I(y^{(i)} = k) (1 - \hat{y}_k^{(i)}) x_j^{(i)} + \lambda w_{k,j} \quad (\because \hat{y}_k^{(i)} = q_k^{(i)}) \\
&= \begin{cases} -(1 - \hat{y}_k^{(i)}) x_j^{(i)} + \lambda w_{k,j}, & \text{if } y^{(i)} = k \\ \lambda w_{k,j}, & \text{if } y^{(i)} \neq k \end{cases}
\end{aligned}$$

因此使用 Mini-Batch-SGD 优化算法时迭代更新公式为：

$$\begin{aligned}
w_{k,j} &\leftarrow w_{k,j} - \eta \left(\frac{1}{m} \sum_{i=1}^m \frac{\partial J(\mathbf{w}; \mathbf{x}^{(i)}; y^{(i)})}{\partial w_{k,j}} \right) \\
w_{k,j} &\leftarrow w_{k,j} - \eta \left(\frac{1}{m} \sum_{i=1}^m \underbrace{I(y^{(i)} = k)}_{p_k^{(i)}} (1 - \hat{y}_k^{(i)}) x_j^{(i)} - \lambda w_{k,j} \right)
\end{aligned}$$

7.2.3 Summary of softmax regression

- **Softmax Regression 中各个类之间不是相互独立的，而是相互影响的。**因为在 logit 转换概率的时候，使用的是 Softmax function，即：

$$\hat{y}_k = q_k = \frac{\exp(z_k)}{\sum_{l=1}^K \exp(z_l)} = \frac{\exp(\sum_{j=1}^d w_{k,j} x_j)}{\sum_{l=1}^K \exp(\sum_{j=1}^d w_{l,j} x_j)}$$

可以看到第 k 类的概率估计里不仅有第 k 类的参数，还有其他所有类的参数（分母），所以其不是类别独立的。

- **指数函数的作用是起到全部整合为正数，同时增强差异对比度**
 - $z_k < 0$ 时， $e^{z_k} \in (0, 1)$ ，有效地将负数转换为了小的正数
 - $z_k > 0$ 时， $e^{z_k} \in (1, +\infty)$ ，且有效地放大了不同 z_k 之间对比程度
- **为了简化多分类问题的计算开销，可将 Softmax 函数转换为 K 个独立的 Logistic 函数**

7.2.4 Reference

- DeepNotes: Classification and Loss Evaluation - Softmax and Cross Entropy Loss

8 Naive Bayes

8.1 Naive Bayes Classification

8.1.1 Model prediction

Naive Bayes 是一个用于多分类的非线性模型预测时，我们需要计算样本属于每一类的概率，然后选择概率最大的那一类作为预测类别。所以样本 \mathbf{x} 属于某一类 k 的概率为：

$$\begin{aligned} P(y = k | \mathbf{x}) &= \frac{P(\mathbf{x}, y = k)}{P(\mathbf{x})} && \text{(条件概率公式)} \\ &= \frac{P(\mathbf{x} | y = k)P(y = k)}{P(\mathbf{x})} && \text{(贝叶斯公式)} \end{aligned}$$

因为分母 $P(\mathbf{x})$ 为归一化项，是一个常数，在比较不同类别概率的时候不起作用，因此计算的时候不必考虑，可以约去，所以实际计算的时候只是：

$$\hat{y} = \arg \max_{k \in K} P(\mathbf{x} | y = k)P(y = k)$$

注意 Naive Bayes Classifier 最关键的一点：**条件独立假设**，即**假设特征之间都是相互独立的，无耦合，互不干扰**（但是实际中这种情况很少很少）。于是有：

$$P(\mathbf{x} | y = k) = \prod_{j=1}^d P(x_j | y = k)$$

所以：**朴素贝叶斯分类器 = 贝叶斯公式 + 条件独立假设**

$$\hat{y} = \arg \max_{k \in K} \left[\prod_{j=1}^d P(x_j | y = k) \right] P(y = k)$$

8.1.2 Model training

Dataset 如下为一个二分类问题的数据集，其中任意 $\mathbf{x}^{(i)} \in \mathbb{R}^3$ ，我们将使用这个例子进行训练过程的解释说明

Example No.	Color	Type	Origin	Stolen?
1	Red	Sports	Domestic	Yes
2	Red	Sports	Domestic	No
3	Red	Sports	Domestic	Yes
4	Yellow	Sports	Domestic	No
5	Yellow	Sports	Imported	Yes
6	Yellow	SUV	Imported	No
7	Yellow	SUV	Imported	Yes
8	Yellow	SUV	Domestic	No
9	Red	SUV	Imported	No
10	Red	Sports	Imported	Yes

Model details 因此在训练中，我们需要计算的内容包括：

- 所有类的 $P(y = k), k \in K$
- 每类每个特征上每个特征取值的 $P(x_j | y = k)$ ，比如：Color 特征里 Red 为一个特征取值 ($x_1 = \text{Red}$)

我们希望估计样本 $\mathbf{x} = (\text{Red}, \text{SUV}, \text{Domestic})$ 是否被 Stolen (这条样本没有出现在训练样本中)，具体计算过程如下：

- $P(y = \text{Yes}) = 0.5$
- $P(y = \text{No}) = 0.5$
- 计算第一个特征 (Color)：
 - $P(x_1 = \text{Red} | y = \text{Yes}) = \frac{3+3 \times 0.5}{5+3} = 0.56$ (注：特征取之的条件概率计算使用了 m-estimates，下同)
 - $P(x_1 = \text{Red} | y = \text{No}) = \frac{2+3 \times 0.5}{5+3} = 0.43$
- 计算第二个特征 (Type)：
 - $P(x_2 = \text{SUV} | y = \text{Yes}) = \frac{1+3 \times 0.5}{5+3} = 0.31$
 - $P(x_2 = \text{SUV} | y = \text{No}) = \frac{3+3 \times 0.5}{5+3} = 0.56$
- 计算第三个特征 (Origin)：
 - $P(x_3 = \text{Domestic} | y = \text{Yes}) = \frac{2+3 \times 0.5}{5+3} = 0.43$
 - $P(x_3 = \text{Domestic} | y = \text{No}) = \frac{3+3 \times 0.5}{5+3} = 0.56$
- 计算最终概率：
 - $P(y = \text{Yes}) \cdot P(x_1 = \text{Red} | y = \text{Yes}) \cdot P(x_2 = \text{SUV} | y = \text{Yes}) \cdot P(x_3 = \text{Domestic} | y = \text{Yes}) = 0.5 \times 0.56 \times 0.31 \times 0.43 = 0.037$
 - $P(y = \text{No}) \cdot P(x_1 = \text{Red} | y = \text{No}) \cdot P(x_2 = \text{SUV} | y = \text{No}) \cdot P(x_3 = \text{Domestic} | y = \text{No}) = 0.5 \times 0.43 \times 0.56 \times 0.56 = 0.069$
- 得出结果：Since $0.037 < 0.069$, our test sample gets classified as "No".

8.1.3 Naive bayes note

- 该模型为非参数模型，因此没有之后目标函数和优化算法的步骤
- 训练和预测同时进行

8.2 Logistic Regression VS Naive Bayes

- Logistic Regression 是**判别模型**，Naive Bayes 是**生成模型**
 - Naive Bayes 在计算 $P(y|x)$ 之间，需要先从训练数据中计算出概率 $P(x|y)$ 以及 $P(y)$
 - Logistic Regression 通过在训练数据上学习直接得到判别函数 $P(y|x)$ ，不需要知道 $P(x|y)$ 以及 $P(y)$
- Naive Bayes **必须建立在特征条件独立假设的基础上**，而 Logistic Regression 则不需要。当然，如果特征之间满足条件独立假设的条件，那么 Logistic Regression 能够取得非常好的效果，如果没有，LR 仍然能够通过调整参数让模型符合数据的分布，从而训练得到在现有数据集下的一个最优模型
- 数据集较小，使用 Naive Bayes；数据集较大，使用 Logistic Regression

9 Graphical Models

9.1 Hidden Markov Models (HMM)

9.1.1 Introduction (未完成)

隐马尔可夫模型 (Hidden Markov Model, HMM) 用于处理时间序列数据。HMM 包含两部分内容：隐藏变量 (hidden variables) 和观测变量 (observation variables)。而 HMM 的能力在于：根据给定已知的观测变量序列，估计对应的隐藏变量序列，并对未来的观测变量序列做预测。

9.1.2 Model formulation

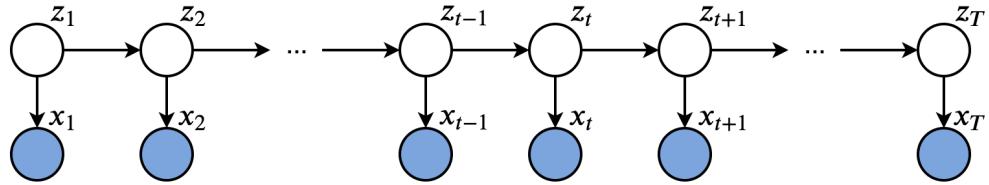


图 38: A Bayesian network specifying conditional independence relations for a hidden markov model.

Component 图中 z_t 为在时刻 t 的隐藏变量， x_t 为对应的观测变量。该模型的关键是隐藏变量之间满足如下条件独立性：给定 z_t 时， z_{t-1} 和 z_{t+1} 条件独立：

$$z_{t-1} \perp z_{t+1} \mid z_t$$

因此该模型的联合概率分布 $p(z_1, \dots, z_T, x_1, \dots, x_T)$ 可以表示如下：

$$p(z_1, \dots, z_T, x_1, \dots, x_T) = p(z_1) \left[\prod_{t=2}^T p(z_t | z_{t-1}) \right] \left[\prod_{t=1}^T p(x_t | z_t) \right]$$

因此该模型被拆解为了三部分：

1. 初始概率模型： $p(z_1)$
2. 转移概率模型： $p(z_t | z_{t-1})$
3. 发射概率模型： $p(x_t | z_t)$

Representation

Learning

9.1.3 Reference

- An Introduction to Hidden Markov Models and Bayesian Networks[?]
- CMU Slides: Hidden Markov Models and Conditional Random Fields

9.2 Conditional Random Fields (CRF)

9.2.1 Model formulation

9.2.2 Reference

- Hanna M.Wallach: Conditional Random Fields
- Edwin Chen: Introduction to Conditional Random Fields
- Eric P.Xing-CMU: Probabilistic Graphical Models

10 Time Series Models

10.1 ARIMA (未完成)

10.1.1 Autoregressive models (AR)

10.1.2 Moving average models (MA)

10.1.3 Autoregressive moving average models (ARMA)

10.2 Online ARIMA Algorithms for Time Series Prediction

10.2.1 Introduction

选自论文Online ARIMA Algorithms for Time Series Prediction[?]

10.3 Prophet (未完成)

10.4 Modeling Long and Short-Term Temporal Patterns with Deep Neural Networks (未完成)

11 Support Vector Machine

11.1 Hard-margin SVM Classification

考虑所有训练样本线性可分的情况，即如图所示，正负样本被完全地分割在了分割超平面的两侧：

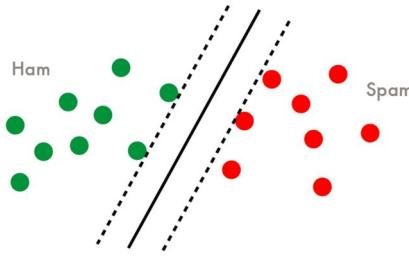


图 39: Linear separability case of SVM

11.1.1 Model prediction

预测阶段之前，我们已经学到了模型的参数 w 和 b ，那么根据预测函数公式：

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

我们可以对测试样本 \mathbf{x} 作出如下预测：

- $f(\mathbf{x}) > 0 \rightarrow \hat{y} = 1$
- $f(\mathbf{x}) < 0 \rightarrow \hat{y} = -1$

11.1.2 Model training

Dataset 训练数据为：

$$\begin{aligned} \mathcal{D}^{n \times d} &= \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N \\ \mathbf{x}^{(i)} &\in \mathbb{R}^d \quad \forall i \in \{1, \dots, n\} \\ y^{(i)} &\in \{-1, +1\} \quad \forall i \in \{1, \dots, n\} \end{aligned}$$

Model details[?] 我们要做的就是训练出参数 w 和 b 。一般意义上，我们需要：

$$f(\mathbf{x}^{(i)}) = \mathbf{w} \cdot \mathbf{x}^{(i)} + b = \begin{cases} > 0 & , \text{ if } y^{(i)} = 1 \\ < 0 & , \text{ if } y^{(i)} = -1 \end{cases}$$

但是为了模型的鲁棒性，我们设置约束条件（最大间隔假设）为：

$$f(\mathbf{x}^{(i)}) = \mathbf{w} \cdot \mathbf{x}^{(i)} + b = \begin{cases} \geq 1 & , \text{ if } y^{(i)} = 1 \\ \leq -1 & , \text{ if } y^{(i)} = -1 \end{cases}$$

合并起来，即：

$$y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) \geq 1, \quad \forall (\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D}$$

而这就是 SVM 训练模型，即一个约束优化模型的约束条件。

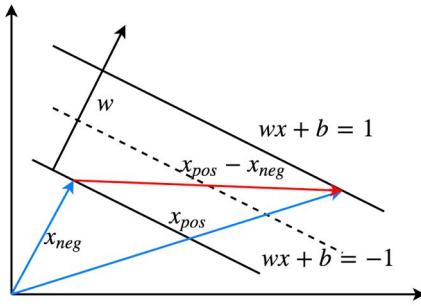


图 40: Width of Streets in SVM

问题：如何计算即“街宽”？

- 在负样本区域“街边”任意取一点，记作 x_-
- 在正样本区域“街边”任意取一点，记作 x_+
- 连线 x_- 和 x_+ ，得到两条街之间任意的一条向量 $(x_+ - x_-)$ （暂且叫做“街线”）
- 两条街之间的距离，实际上就是街线在法向量 w 方向上的投影，即：

$$\begin{aligned} \text{width} &= \|(x_+ - x_-)\| \cdot \cos \theta \\ &= \|(x_+ - x_-)\| \cdot \frac{(x_+ - x_-) \cdot w}{\|(x_+ - x_-)\| \cdot \|w\|} \\ &= \frac{(x_+ - x_-) \cdot w}{\|w\|} \\ &= \frac{w \cdot x_+}{\|w\|} - \frac{w \cdot x_-}{\|w\|} \end{aligned}$$

- 街边的点满足条件： $y_i(w \cdot x_i + b) = 1$ ，因此：

- 对 x_+ : 因为 $y_i = 1$ ，所以 $w \cdot x_+ = 1 - b$
- 对 x_- : 因为 $y_i = -1$ ，所以 $w \cdot x_- = -1 - b$

- 故而街道的距离可化简为：

$$\begin{aligned} \text{width} &= \frac{1 - b}{\|w\|} - \frac{-(1 + b)}{\|w\|} \\ &= \frac{1 - b + 1 + b}{\|w\|} \\ &= \frac{2}{\|w\|} \end{aligned}$$

至此，优化目标和约束条件全有了，那么我们的训练模型可以建模为如下：

$$\begin{aligned} \max \quad & \frac{2}{\|w\|} \\ \text{s.t. } & y^{(i)} \cdot (w \cdot x^{(i)} + b) \geq 1 \quad \forall (x^{(i)}, y^{(i)}) \in \mathcal{D} \end{aligned}$$

将目标函数进行等价变换，得到如下**最终原始优化模型 (primal problem)**：

$$\begin{aligned} & \min \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s.t. } & y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) \geq 1 \quad \forall (\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D} \end{aligned}$$

这个原始模型有以下特点：

- 目标函数是关于 \mathbf{w} 的凸函数
- 约束条件是关于 \mathbf{w} 的线性函数

那么其对偶问题的最优解一定等于原始问题的最优解，如下介绍 SVM 原始模型的拉格朗日对偶模型。

SVM 对偶模型 SVM 原始模型的拉格朗日函数为：

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^N \alpha^{(i)} (1 - y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b))$$

注意：原始模型的约束条件为 \geq ，那么拉格朗日函数里为 $-$ ，即正好相反。根据 KKT 条件，即：

$$\left\{ \begin{array}{l} \nabla_{\mathbf{w}} = 0, \\ \nabla_b = 0, \\ 1 - y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) \leq 0, \quad \forall (\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D} \\ \alpha^{(i)} \geq 0, \quad \forall i \in \{1, \dots, n\} \\ \alpha^{(i)} (1 - y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b)) = 0, \quad \forall i \in \{1, \dots, n\} \end{array} \right.$$

那么根据第一个约束有：

$$\begin{aligned} \nabla_{\mathbf{w}} &= \mathbf{w} + \sum_{i=1}^N \alpha^{(i)} (-y^{(i)} \mathbf{x}^{(i)}) \\ &= \mathbf{w} - \sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} = 0 \\ \mathbf{w} &= \sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \end{aligned}$$

根据第二个约束有：

$$\begin{aligned} \nabla_b &= \sum_{i=1}^N \alpha^{(i)} (-y^{(i)}) = 0 \\ \sum_{i=1}^N \alpha^{(i)} y^{(i)} &= 0 \end{aligned}$$

将 $\mathbf{w} = \sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}$ 和 $\sum_{i=1}^N \alpha^{(i)} y^{(i)} = 0$ 代入拉格朗日函数（还是一个最小化问题）中，可得：

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w} \cdot \mathbf{w} + \sum_{i=1}^N \alpha^{(i)} - \sum_{i=1}^N \alpha^{(i)} y^{(i)} (\mathbf{w} \cdot \mathbf{x}^{(i)}) - b \sum_{i=1}^N \alpha^{(i)} y^{(i)}$$

$$\begin{aligned}
&= \frac{1}{2} \left(\sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \right)^2 + \sum_{i=1}^N \alpha^{(i)} - \sum_{i=1}^N \alpha^{(i)} y^{(i)} \left(\left(\sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \right) \cdot \mathbf{x}^{(i)} \right) \\
&= \underbrace{\frac{1}{2} \left(\sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \right) \left(\sum_{j=1}^n \alpha^{(j)} y^{(j)} \mathbf{x}^{(j)} \right) + \sum_{i=1}^N \alpha^{(i)} - \sum_{i=1}^N \alpha^{(i)} y^{(i)} \left(\left(\sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \right) \cdot \mathbf{x}^{(i)} \right)}_{\text{用两个索引表示, 之后可以交叉}} \\
&= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} + \sum_{i=1}^N \alpha^{(i)} - \underbrace{\sum_{i=1}^N \alpha^{(i)} y^{(i)} \left(\left(\sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \right) \cdot \mathbf{x}^{(i)} \right)}_{\text{对此项进行变换}} \\
&= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} + \sum_{i=1}^N \alpha^{(i)} - \underbrace{\sum_{i=1}^N \alpha^{(i)} y^{(i)} \left(\sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \right) \cdot \mathbf{x}^{(i)}}_{\text{都是在一起的乘法, 把括号抹掉}} \\
&= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} + \sum_{i=1}^N \alpha^{(i)} - \underbrace{\sum_{i=1}^N \alpha_i y_i \mathbf{x}_i \left(\sum_{j=1}^n \alpha_j y_j \mathbf{x}_j \right)}_{\text{把 } \mathbf{x}_i \text{ 提到前面}} \\
&= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} + \sum_{i=1}^N \alpha^{(i)} - \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} \\
&= \sum_{i=1}^N \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} = Q(\boldsymbol{\alpha})
\end{aligned}$$

问题：既然 $\alpha^{(i)} (1 - y^{(i)} (\mathbf{w} \cdot \mathbf{x}^{(i)} + b)) = 0, \forall i \in \{1, \dots, n\}$, 为何不直接在拉格朗日函数中约掉第二项, 还拼死拼活地展开? 因为其中含有拉格朗日乘子, 我们要将拉格朗日函数表示为乘子的函数, 因此不能舍去。

Loss function 因此 SVM 对偶模型为 :

$$\begin{aligned}
\max Q(\boldsymbol{\alpha}) &= \sum_{i=1}^N \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} \\
\text{s.t. } &\sum_{i=1}^N \alpha^{(i)} y^{(i)} = 0, \\
&\alpha^{(i)} \geq 0, \forall i \in \{1, \dots, n\}
\end{aligned}$$

注意这里的约束条件是 KKT 条件里第二个和第四个, 至于为什么没有第一个, 第三个和第五个, 因为这三个其中都有 \mathbf{w} , 而对偶里面没有 \mathbf{w} 。

Optimization 这样解出了最优的 $\boldsymbol{\alpha}^*$ 之后, 反带回去即可得到参数 \mathbf{w} 和 b 的最优解, 即 :

$$\begin{aligned}
\mathbf{w}^* &= \sum_{i=1}^N (\alpha^{(i)})^* y^{(i)} \mathbf{x}^{(i)} \\
b^* &= 1 - \mathbf{w}^* \cdot \mathbf{x}^{(i)}, \quad (\mathbf{x}^{(i)} \text{ 为某一个正的支持向量})
\end{aligned}$$

关于 w^* 以及 b^* 求解的解释说明 根据 KKT 条件的第五个条件，即：

$$\alpha^{(i)}(1 - y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b)) = 0, \quad \forall i \in \{1, \dots, n\}$$

那么我们可以得知，对于所有的训练样本 $(\mathbf{x}^{(i)}, y^{(i)})$ ：

- 要么 $\alpha^{(i)} = 0$ ，且 $1 - y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) \neq 0$ （非支持向量）
- 要么 $y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) = 1$ ，且 $\alpha^{(i)} > 0$ （支持向量）

再看看参数 w 的最优解公式：

$$\mathbf{w}^* = \sum_{i=1}^N (\alpha^{(i)})^* y^{(i)} \mathbf{x}^{(i)}$$

也就是说：那么非支持向量点，并不影响 w^* （因为对应 $\alpha^{(i)} = 0$ ），参数 w 的最优解仅仅取决于所有的支持向量点。在计算 b^* 时，我们就仅拿一个正支持向量进行计算就好，即：

$$\begin{aligned} \mathbf{w}^* \cdot \mathbf{x}^{\text{support+}} + b &= 1 \\ b^* &= 1 - \mathbf{w}^* \cdot \mathbf{x}^{\text{support+}} \end{aligned}$$

所以得到的模型判别函数为：

$$f(\mathbf{x}) = \left(\sum_{i=1}^N (\alpha^{(i)})^* y^{(i)} \mathbf{x}^{(i)} \right) \cdot \mathbf{x} + b^*$$

11.1.3 Summary of hard-margin svm

- SVM 本质是线性分类器，实际用的判别函数为线性函数 $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$ ，只不过训练过程建模为一个约束优化模型。
- SVM 的样本类别标签表示不同于一般模型。Logistic Regression 模型里，样本的类别标签 $y \in \{0, 1\}$ ，而 SVM 里样本类别标签 $y \in \{-1, 1\}$ 。

11.2 Soft-margin SVM Classification

之前 Hard-margin SVM Classification 的推导都是基于所有训练样本线性可分的假设下进行的，但实际中这种情况几乎不存在，因此我们想在容忍一些误分类样本的情况下最大化间隔。具体的方法是：为每个训练样本 (\mathbf{x}_i, y_i) 引入一个松弛变量 ξ_i ，用于度量错误分类的程度。

11.2.1 Model prediction

预测函数同上：

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

$$\hat{y} = \begin{cases} -1, & \text{if } f(\mathbf{x}) > 0 \\ +1, & \text{if } f(\mathbf{x}) < 0 \end{cases}$$

其中 \mathbf{w} 和 b 为待学习参数

11.2.2 Model training of dual formulation

Dataset 训练数据为：

$$\mathcal{D}^{n \times d} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$$

$$\mathbf{x}^{(i)} \in \mathbb{R}^d, \quad \forall i \in \{1, \dots, n\}$$

$$y^{(i)} \in \{-1, +1\} \quad \forall i \in \{1, \dots, n\}$$

Model details

原始模型

$$\begin{aligned} \min Q_1(\mathbf{w}, b, \xi) &= \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi^{(i)} \\ \text{s.t. } y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) &\geq 1 - \xi^{(i)}, \quad \forall (\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D} \\ \xi^{(i)} &\geq 0, \quad \forall i \in \{1, \dots, n\}. \end{aligned}$$

对偶模型求解过程如下： 原始模型的拉格朗日函数为：

$$J(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi^{(i)} + \sum_{i=1}^N \alpha^{(i)} \left(1 - \xi^{(i)} - y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) \right) - \sum_{i=1}^N \beta^{(i)} \xi^{(i)}$$

根据 KKT 条件，即：

$$\left\{ \begin{array}{l} \nabla_{\mathbf{w}} = 0 \\ \nabla_b = 0 \\ \nabla_{\xi} = 0 \\ 1 - \xi^{(i)} - y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) \leq 0, \quad \forall (\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D} \\ \xi^{(i)} \geq 0, \quad \forall i \in \{1, \dots, n\} \\ \alpha^{(i)} \geq 0, \quad \forall i \in \{1, \dots, n\} \\ \beta^{(i)} \geq 0, \quad \forall i \in \{1, \dots, n\} \\ \alpha^{(i)} (1 - \xi^{(i)} - y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b)) = 0, \quad \forall (\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D} \\ \beta^{(i)} \xi^{(i)} = 0 \quad \forall i \in \{1, \dots, n\} \end{array} \right.$$

解第一个约束，可得：

$$\nabla_{\mathbf{w}} = \mathbf{w} - \sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} = 0$$

解第二个约束，可得：

$$\nabla_b = - \sum_{i=1}^N \alpha^{(i)} y^{(i)} = 0$$

解第三个约束，可得：

$$\nabla_{\xi^{(i)}} = C - \alpha^{(i)} - \beta^{(i)} = C - (\alpha^{(i)} + \beta^{(i)}) = 0, \quad \forall i \in \{1, \dots, n\}$$

将解代入拉格朗日函数中，可得：

$$\begin{aligned} J(\mathbf{w}, b, \xi, \alpha, \beta) &= \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^N \xi^{(i)} + \sum_{i=1}^N \alpha^{(i)} \left(1 - \xi^{(i)} - y^{(i)} (\mathbf{w} \cdot \mathbf{x}^{(i)} + b) \right) - \sum_{i=1}^N \beta^{(i)} \xi^{(i)} \\ &= \frac{1}{2} \sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \sum_{j=1}^n \alpha^{(j)} y^{(j)} \mathbf{x}^{(j)} + C \sum_{i=1}^N \xi^{(i)} + \sum_{i=1}^N \alpha^{(i)} - \sum_{i=1}^N \alpha^{(i)} \xi^{(i)} \\ &\quad - \sum_{i=1}^N \alpha^{(i)} y^{(i)} (\mathbf{w} \cdot \mathbf{x}^{(i)}) - b \underbrace{\sum_{i=1}^N \alpha^{(i)} y^{(i)}}_{\text{为 } 0, \text{ 约掉}} - \sum_{i=1}^N \beta^{(i)} \xi^{(i)} \\ &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} + C \sum_{i=1}^N \xi^{(i)} + \sum_{i=1}^N \alpha^{(i)} - \sum_{i=1}^N (\alpha^{(i)} + \beta^{(i)}) \xi^{(i)} \\ &\quad - \sum_{i=1}^N \alpha^{(i)} y^{(i)} (\mathbf{w} \cdot \mathbf{x}^{(i)}) \\ &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} + \sum_{i=1}^N \alpha^{(i)} + \sum_{i=1}^N \underbrace{(C - (\alpha^{(i)} + \beta^{(i)})) \xi^{(i)}}_{\text{为 } 0, \text{ 约掉}} \\ &\quad - \sum_{i=1}^N \alpha^{(i)} y^{(i)} \left(\left(\sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \right) \mathbf{x}^{(i)} \right) \\ &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} + \sum_{i=1}^N \alpha^{(i)} - \sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \left(\sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \right) \\ &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} + \sum_{i=1}^N \alpha^{(i)} - \sum_{i=1}^N \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \sum_{j=1}^n \alpha^{(j)} y^{(j)} \mathbf{x}^{(j)} \\ &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} + \sum_{i=1}^N \alpha^{(i)} - \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} \\ &= \sum_{i=1}^N \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)} = Q_2(\alpha) \end{aligned}$$

对偶模型

$$\max Q_2(\alpha) = \sum_{i=1}^N \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)} \mathbf{x}^{(j)}$$

$$\text{s.t. } \sum_{i=1}^N \alpha^{(i)} y^{(i)} = 0,$$

$$0 \leq \alpha^{(i)} \leq C, \quad \forall i \in \{1, \dots, n\}$$

该模型可以解得 α^* ，然后同上代入得到 w^* 以及 b^* .

注意

- 注意在 Soft-margin 的对偶问题中，松弛变量 $\xi^{(i)}$ 和其对应的拉格朗日乘子 $\beta^{(i)}$ 都不见了，该 Soft-margin 对偶模型和 Hard-margin 对偶模型唯一的区别就是乘子 $\alpha^{(i)}$ 的约束条件在 Soft-margin 中变得更严苛.

11.2.3 Model training of hinge loss

Hinge loss $l(z) = \max(0, 1 - z)$ 是一种损失函数，又称作 Max-margin objective，最著名的应用就是作为 SVM 的目标函数。

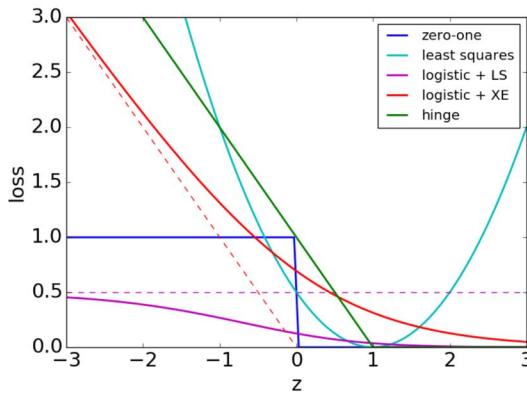


图 41: The hinge loss and other losses

- Binary classification:

$$l(\hat{y}) = \max(0, 1 - y \cdot \hat{y})$$

其中：

- $y \in \{-1, +1\}$ 为真实值
- $\hat{y} \in (-1, +1)$ 为预测值

含义：

- 正确分类的损失 $l \in [0, 1)$
- 错误分类的损失 $l \in (1, +\infty)$
- $\hat{y} \in [-1, +1]$ 满足就可以了，不鼓励 $|\hat{y}| > 1$ ，即不鼓励模型**过分自信**。譬如： $y = +1$ ，而预测 $\hat{y} = +1000$ 时，虽然 $1 - y \cdot \hat{y} = 1 - 1000 = -999$ ，但是 $l = \max(0, -999) = 0$ ，因此不会得到任何奖励，即 reward = $-l = -0 = 0$ 。

Model details 两种求 Soft-margin SVM 模型的方式：

- 原始模型 \Rightarrow 对偶模型
- 原始模型 \Rightarrow Hinge loss

原始模型如下：

$$\begin{aligned} \min Q_1(\mathbf{w}, b, \xi) &= \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi^{(i)} \\ \text{s.t. } y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) &\geq 1 - \xi^{(i)}, \quad \forall (\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D} \\ \xi^{(i)} &\geq 0, \quad \forall i \in \{1, \dots, n\}. \end{aligned}$$

将约束进行变换，得：

$$\begin{aligned} \xi^{(i)} &\geq 1 - y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b), \quad \forall i \in \{1, \dots, n\}. \\ \xi^{(i)} &\geq 0, \quad \forall i \in \{1, \dots, n\}. \\ \therefore \xi^{(i)} &\geq \max(0, 1 - y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b)) \end{aligned}$$

Loss function 因此优化损失函数可以表示为：

$$\begin{aligned} \min J(\mathbf{w}; b) &= \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \max(0, 1 - y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b)) \\ \Rightarrow \min J(\mathbf{w}; b) &= \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \max(0, 1 - z^{(i)}(\mathbf{w}; b)) \\ \Rightarrow \min J(\mathbf{w}; b) &= \underbrace{\frac{1}{2} \|\mathbf{w}\|^2}_{\text{l2-regularization}} + \underbrace{C \sum_{i=1}^N \mathcal{L}_{\text{hinge}}(z^{(i)})}_{\text{sum of hinge loss}} \end{aligned}$$

好处

- 转换为了明确的 loss function，便于使用机器学习框架化的程式进行学习训练
- 可以在线学习

Optimization 求解 $\frac{\partial J}{\partial w_j}$ 和 $\frac{\partial J}{\partial b}$ 即可：

- $\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial l} \frac{\partial l}{\partial z} \frac{\partial z}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_j}$
- $\frac{\partial J}{\partial b} = \frac{\partial J}{\partial l} \frac{\partial l}{\partial z} \frac{\partial z}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b}$

其中：

$$\frac{\partial l(z)}{\partial z} = \begin{cases} 0, & \text{if } z \geq 1; \\ -1, & \text{if } z < 1. \end{cases}$$

11.3 Kernel Trick for Nonlinear Classification

Introduction SVM 本质是个线性模型（超平面），但是如果训练数据不是线性可分的，那么就没有一个直接的超平面可以分离正负样本。因此为了学习一个非线性函数（nonlinear function），线性 SVM（Linear SVM）需要扩展到非线性 SVM（Nonlinear SVM），用以分类那些线性不可分的数据。

确定一个非线性 SVM 的分类函数（判定函数）包含两个步骤：

- 所有训练样本的特征向量需要转换为可以被线性分离的高维特征向量
- 在高维空间中找到一个可以将训练数据线性分离的分割超平面

这样一来，训练出的分割超平面在转换特征空间（transformed feature space）内是线性的，但是在原始特征空间（original feature space）内是非线性的。

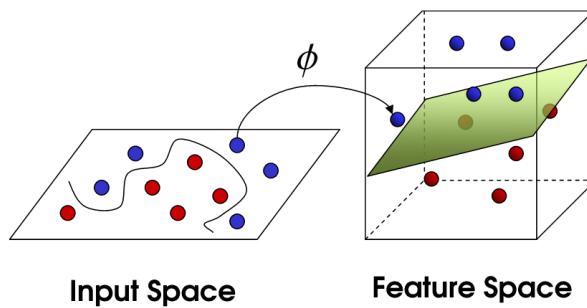


图 42: Feature Space Transformation using Kernel Method

我们用 $\phi(\cdot)$ 表示从原始特征空间到高维特征空间的非线性映射，那么分离边界可以表示为：

$$\mathbf{w} \cdot \phi(\mathbf{x}) + b = 0$$

对偶优化模型变为：

$$\begin{aligned} \max Q(\alpha) &= \sum_{i=1}^N \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)}) \\ \text{s.t. } & \sum_{i=1}^N \alpha^{(i)} y^{(i)} = 0, \\ & 0 \leq \alpha^{(i)} \leq C, \quad \forall i \in \{1, \dots, n\} \end{aligned}$$

存在问题

- 一个有效的非线性映射 $\phi(\cdot)$ 非常难找到
- 直接计算 $\phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)})$ 非常复杂，受制于维度的问题。

解决方法 通过核技巧（kernel trick）解决，即：

$$K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u}) \cdot \phi(\mathbf{v})$$

因此核函数是作用在原始特征空间，用以替换在高维特征空间的成对儿特征向量的内积。因此上述对偶模型转换为：

$$\begin{aligned} \max Q(\boldsymbol{\alpha}) &= \sum_{i=1}^N \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^n \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \\ \text{s.t. } & \sum_{i=1}^N \alpha^{(i)} y^{(i)} = 0, \\ & 0 \leq \alpha^{(i)} \leq C, \quad \forall i \in \{1, \dots, n\} \end{aligned}$$

所以最终的解为：

$$\begin{aligned} f(\mathbf{x}) &= \left(\sum_{i=1}^N (\alpha^{(i)})^* y^{(i)} \phi(\mathbf{x}^{(i)}) \right) \cdot \phi(\mathbf{x}) + b^* \\ &= \sum_{i=1}^N (\alpha^{(i)})^* y^{(i)} \phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}) + b^* \\ &= \sum_{i=1}^N (\alpha^{(i)})^* y^{(i)} K(\mathbf{x}^{(i)}, \mathbf{x}) + b^* \end{aligned}$$

12 Decision Tree

12.1 Classification Tree

决策树（分类树）依据划分准则不同，可分为三类：

- ID3：信息增益

$$\text{Gain}(\mathcal{D}, a) = \text{Ent}(\mathcal{D}) - \sum_{v \in V_a} \frac{|\mathcal{D}_{a=v}|}{|\mathcal{D}|} \text{Ent}(\mathcal{D}_{a=v})$$

$$\text{Ent}(\mathcal{D}) = - \sum_{k=1}^K p_k \log p_k \quad (|K| \text{ is the total number of classes of labels})$$

- C4.5：信息增益率

$$\text{Gain-ratio}(\mathcal{D}, a) = \frac{\text{Gain}(\mathcal{D}, a)}{\text{IV}(a)}$$

$$\text{IV}(a) = - \sum_{v \in V_a} \frac{|\mathcal{D}_{a=v}|}{|\mathcal{D}|} \log \frac{|\mathcal{D}_{a=v}|}{|\mathcal{D}|}$$

- Classification Tree：基尼指数

$$\text{Gini-index}(\mathcal{D}, a) = \sum_{v \in V_a} \frac{|\mathcal{D}_{a=v}|}{|\mathcal{D}|} \text{Gini}(\mathcal{D}_{a=v})$$

$$\text{Gini}(\mathcal{D}) = 1 - \sum_{k=1}^K p_k^2 \quad (|K| \text{ is the total number of classes of labels})$$

12.1.1 Model prediction

如图所示， x 根据树的路径找到对应的叶子结点，返回叶子结点对应的预测输出。

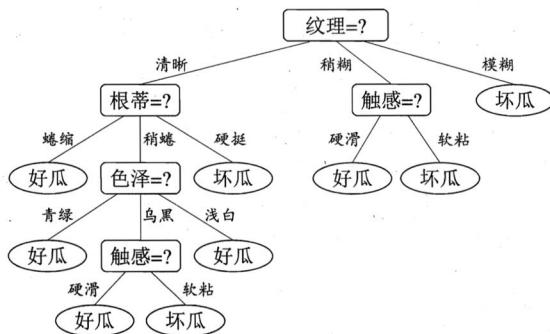


图 43: Decision Tree ID3

12.1.2 Model training

Dataset 考虑如下数据集 \mathcal{D} :

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜 ?
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	否
10	青绿	硬挺	清脆	清晰	平坦	软粘	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	否

我们观测到所有特征上的所有可能取值如下：

- 色泽 : {青绿, 乌黑, 浅白}
- 根蒂 : {蜷缩, 稍蜷, 硬挺}
- 敲声 : {浊响, 沉闷, 清脆}
- 纹理 : {清晰, 稍糊, 模糊}
- 脐部 : {凹陷, 稍凹, 平坦}
- 触感 : {硬滑, 软粘}

现在需要计算数据集在每个 (属性, 候选值) 的划分准则取值。

Inference (e.g. ID3)

- 计算原始数据集 \mathcal{D} 上的信息熵：

$$\begin{aligned}
 \text{Ent}(\mathcal{D}) &= - \sum_{k=1}^K p_k \log p_k \\
 &= - \sum_{k=1}^2 p_k \log p_k \\
 &= - \left(\frac{8}{17} \log \frac{8}{17} + \frac{9}{17} \log \frac{9}{17} \right) = 0.998
 \end{aligned}$$

- 计算以属性“色泽”作为分割点的信息增益 $\text{Gain}(\mathcal{D}, \text{色泽})$ ：

– 1. 计算 $\mathcal{D}_{\text{色泽}=\text{青绿}}$ 上的信息熵：

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜？
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	是
10	青绿	硬挺	清脆	清晰	平坦	软粘	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	否

$$\text{Ent}(\mathcal{D}_{\text{色泽}=\text{青绿}}) = -\left(\frac{3}{6} \log \frac{3}{6} + \frac{3}{6} \log \frac{3}{6}\right) = 1.0$$

– 2. 计算 $\mathcal{D}_{\text{色泽}=\text{乌黑}}$ 上的信息熵：

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜？
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	否

$$\text{Ent}(\mathcal{D}_{\text{色泽}=\text{乌黑}}) = -\left(\frac{4}{6} \log \frac{4}{6} + \frac{2}{6} \log \frac{2}{6}\right) = 0.918$$

– 3. 计算 $\mathcal{D}_{\text{色泽}=\text{浅白}}$ 上的信息熵：

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜？
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	是
11	浅白	硬挺	清脆	模糊	平坦	硬滑	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	否

$$\text{Ent}(\mathcal{D}_{\text{色泽}=\text{浅白}}) = -\left(\frac{1}{5} \log \frac{1}{5} + \frac{4}{5} \log \frac{4}{5}\right) = 0.722$$

– 4. 计算信息增益：

$$\begin{aligned} \text{Gain}(\mathcal{D}, \text{色泽}) &= \text{Ent}(\mathcal{D}) - \sum_{v=1}^V \frac{|\mathcal{D}_{\text{色泽}=v}|}{|\mathcal{D}|} \text{Ent}(\mathcal{D}_{\text{色泽}=v}) \\ &= 0.998 - \left(\frac{6}{17} \times 1.000 + \frac{6}{17} \times 0.918 + \frac{5}{17} \times 0.722\right) \\ &= 0.109 \end{aligned}$$

- 计算以属性“根蒂”作为分割点的信息增益 $\text{Gain}(\mathcal{D}, \text{根蒂})$:

$$\text{Gain}(\mathcal{D}, \text{根蒂}) = 0.143$$

- 计算以属性“敲声”作为分割点的信息增益 $\text{Gain}(\mathcal{D}, \text{敲声})$:

$$\text{Gain}(\mathcal{D}, \text{敲声}) = 0.141$$

- 计算以属性“纹理”作为分割点的信息增益 $\text{Gain}(\mathcal{D}, \text{纹理})$:

$$\text{Gain}(\mathcal{D}, \text{纹理}) = 0.381$$

- 计算以属性“脐部”作为分割点的信息增益 $\text{Gain}(\mathcal{D}, \text{脐部})$:

$$\text{Gain}(\mathcal{D}, \text{脐部}) = 0.289$$

- 计算以属性“触感”作为分割点的信息增益 $\text{Gain}(\mathcal{D}, \text{触感})$:

$$\text{Gain}(\mathcal{D}, \text{触感}) = 0.006$$

- 选择最有划分属性：纹理，并分割数据集：

– $\mathcal{D}_{\text{纹理}} = \text{清晰}$:

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜？
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	是
10	青绿	硬挺	清脆	清晰	平坦	软粘	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	否

– $\mathcal{D}_{\text{纹理}} = \text{稍糊}$:

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜？
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	否

– $\mathcal{D}_{\text{纹理}} = \text{模糊}$:

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜？
11	浅白	硬挺	清脆	模糊	平坦	硬滑	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	否

- 依次对 $\mathcal{D}_{\text{纹理}} = \text{清晰}$, $\mathcal{D}_{\text{纹理}} = \text{稍糊}$ 和 $\mathcal{D}_{\text{纹理}} = \text{模糊}$ 重复上述对 \mathcal{D} 的操作...

注意：信息熵描述的是一个概率分布的期望信息量，即**一个概率分布的不确定性：数值越大，不确定性越高**。决策树中希望通过属性对原始数据进行划分从而降低信息熵，因此一般划分前的信息熵大于划分后的信息熵，因此信息增益是正数。

ID3 \Rightarrow C4.5 使用 Information Gain 作为划分准则的时候，倾向于选择值域 $|V_a|$ 多的属性 a ，然而很多时候，这样的划分方式会使得模型陷入过拟合，不具有泛化能力，使得模型无法对新样本作出有效预测。因此引入了信息增益率。

注意：C4.5 不是直接使用 Gain-ratio 作为选择划分属性的依据 Gain-ratio 倾向选择值域 $|V_a|$ 较少的属性 a ，因此 C4.5 算法不是直接使用 Gain-ratio 作为选择划分属性的依据，而是使用**启发式方法**，譬如：先从候选划分属性中找出 Gain 大于平均水平的属性，然后再从中选择 Gain-ratio 最高的。

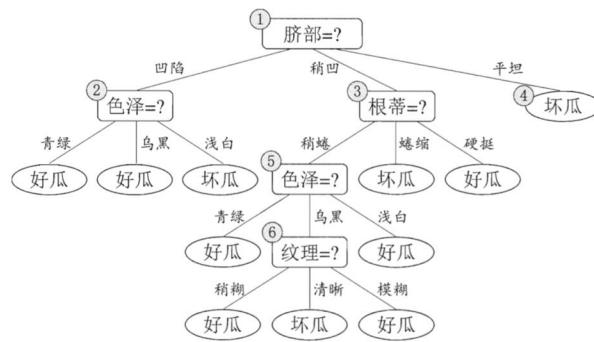


图 44: Decision tree with pre-pruning

问题：如何处理连续型特征（属性）？ 直观的想法是：将连续型属性转化为离散型属性。假设数据集 \mathcal{D} 中连续属性 a 包含了 n 个不同的取值，则：

- 从小到大排序 n 个取值： $\{v_1, v_2, \dots, v_n\}$
- 构造属性 a 的候选划分点集合： $T_a = \left\{ \frac{v_i + v_{i+1}}{2} \mid 1 \leq i \leq n - 1 \right\}$
- 根据候选划分点集合中的每个元素 $t \in T_a$ 对样本进行二分： $\mathcal{D}_{a \leq t}$ 和 $\mathcal{D}_{a > t}$
- 计算 Gain (或者 Gain-ratio...) :

$$\begin{aligned} \text{Gain}(\mathcal{D}, a) &= \max_{t \in T_a} \text{Gain}(\mathcal{D}, a, t) \\ &= \max_{t \in T_a} \text{Ent}(\mathcal{D}) - \sum_{\lambda \in \{+, -\}} \frac{|\mathcal{D}_t^\lambda|}{|\mathcal{D}|} \text{Ent}(\mathcal{D}_t^\lambda) \end{aligned}$$

譬如：全部训练样本中，某连续型属性 a 的所有可能取值为： $\{3, 7, 1, 9\}$ ，那么排序后为 $\{1, 3, 7, 9\}$ ，然后构造候选划分点集合为 $\{2, 5, 8\}\dots$

12.1.3 Model pruning

剪枝 (pruning) 分为两种：

- Pre-pruning: 在决策树生成过程中，对每个节点在划分前进行估计，若当前节点的划分不能带来决策树泛化性能提升 (e.g. 验证集上计算 Accuracy)，则停止划分并将当前节点作为叶节点。
- Post-pruning: 先利用训练集完整地生成一棵决策树，然后自底向上地对非叶子结点 v 进行考察，若将以该结点 v 为根结点的子树替换为叶结点，并能带来决策树泛化性能 (e.g. 验证集上计算 Accuracy) 的提升，则将该子树替换为叶子结点。



图 45: Decision tree without pruning

注意：决策树泛化性能的评价一定得在验证集上进行。优缺点对比：

- Pre-pruning:
 - 优点：时间开销相对于 Post-pruning 更小（因为是在训练树的过程中剪枝）
 - 缺点：容易欠拟合
- Post-pruning:
 - 优点：相对于 Pre-pruning 的决策树保留了更多的分支，因此欠拟合风险更小，泛化性能往往优于 Pre-pruning 决策树
 - 缺点：时间开销更大

12.2 Regression Tree

12.2.1 Model training

Algorithm 11: Regression Tree Algorithm

Input: Training dataset $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$, where $\mathbf{x}^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$.

Output: The hypothesis $h(\mathbf{x})$.

```

1 for  $j = 1, \dots, d$  do
2   for each candidate split value of feature  $j$ :  $v_{j,l} \in \mathbb{R}$  do
3     Split the training dataset into two parts:  $I_< = \{i : x_j^{(i)} < v_{j,l}\}$  and  $I_> = \{i : x_j^{(i)} \geq v_{j,l}\}$ 
4     Estimate the prediction values of two parts:  $\hat{y}_< = \frac{\sum_{i \in I_<} y^{(i)}}{|I_<|}$  and  $\hat{y}_> = \frac{\sum_{i \in I_>} y^{(i)}}{|I_>|}$ 
5     Quality of split  $v_{j,l}$  is measured by the squared loss:  $\sum_{i \in I_<} (y^{(i)} - \hat{y}_<)^2 + \sum_{i \in I_>} (y^{(i)} - \hat{y}_>)^2$ 
6   end
7 end
8 Choose the split point with minimal loss
9 Recurse on both children, with  $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i \in I_<}$  and  $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i \in I_>}$ .

```

13 Ensemble Learning

13.1 Random Forest

13.1.1 Model formulation

Algorithm 12: Random Forest Regression Algorithm[?]

Input: Training dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^N$, where $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$;

Column sample rate $\alpha \in (0, 1]$;

Number of base tree model (CART): T.

Output: The final hypothesis $G(x) = \frac{1}{T} \sum_{t=1}^T h_t(x)$.

- 1 Initialize the set of weak regressors as $H = \{\}$
 - 2 **for** $t = 1, \dots, T$ **do**
 - 3 **Sampling with replacement** n training samples from raw training dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^N$, generate new training dataset: $\mathcal{X}_t \in \mathbb{R}^{n \times d}$, $\mathcal{Y}_t \in \mathbb{R}^n$
 - 4 Randomly feature sample with ($d' = \alpha d$) specific features for all samples in \mathcal{X}_t , generate new training samples $\mathcal{X}'_t \in \mathbb{R}^{n \times d'}$
 - 5 Train the weak tree regressor $h_t(x)$, providing it with the training dataset $(\mathcal{X}'_t, \mathcal{Y}_t)$
 - 6 Append weak regressor $h_t(x)$ to H .
 - 7 **end**
-

行采样 采用**bootstrap**的方法，即采样次数等同于样本个数 n ，但是每次采样都是有放回的采样。类似“盲人摸象”。

列采样 指定一个常数 $d' \leq d$ ，在一次树模型的训练过程中，随机地从 d 维特征中选择 d' 维，切片数据集形成维度缩减的数据集用于训练。

不剪枝 一般训练随机森林的树模型不需要剪纸操作，因为对于单一的树模型，剪纸操作是为了防止过拟合，但是随机森林里，因为最后要平均，减小了 Variance，所以无需剪枝减小 Variance。

13.1.2 Random forest classification

预测值为“多数组票”

13.1.3 Random forest regression

预测值为“多值均值”

13.2 Adaboost

13.2.1 Adaboost for binary classification

Algorithm 13: Adaboost for Binary Classification Task

Input: Training dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^N$, where $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \{-1, +1\}$;

Weak hypothesis $h(x)$;

Number of weak hypothesis T.

Output: The final hypothesis $G(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$.

1 Initialize samples weight distribution: $\omega_1^{(i)} = \frac{1}{n}$, for all $i \in \{1, \dots, n\}$

2 **for** $t = 1, \dots, T$ **do**

3 Train the weak learner $h_t(x)$, providing it with the distribution ω_t

4 Get weak hypothesis $h_t(x) : \mathbb{R}^d \rightarrow \{-1, +1\}$ with error:

$$\epsilon_t = \Pr_{\omega_t}(h_t(x^{(i)}) \neq y^{(i)}) = \sum_{i=1}^N I(h_t(x^{(i)}) \neq y^{(i)}) \cdot \omega_t^{(i)}$$

5 Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

6 Update sample weight distribution over all training samples:

$$\omega_{t+1}^{(i)} = \begin{cases} \frac{\omega_t^{(i)}}{Z_t} \times e^{-\alpha_t} & \text{if } h_t(x^{(i)}) = y^{(i)} \\ \frac{\omega_t^{(i)}}{Z_t} \times e^{\alpha_t} & \text{if } h_t(x^{(i)}) \neq y^{(i)} \end{cases}$$

where Z_t is a normalization factor, chosen so that ω_{t+1} will be a distribution, namely:

$$Z_t = \sum_{i=1}^N \omega_t^{(i)} \exp(-\alpha_t y^{(i)} h_t(x^{(i)}))$$

7 **end**

可见在 Adaboost 的学习过程中，我们先初始化样本权重分布，然后开始迭代，在每一步迭代 t 中，我们需要做四件事情：

1. 在当前样本权重分布 ω_t 的情况下训练一个弱学习器 $h_t(x)$
2. 计算 $h_t(x)$ 在分布 ω_t 下的训练集 $\{(x^{(i)}, y^{(i)})\}_{i=1}^N$ 上的误差率 ϵ_t
3. 计算系数 α_t
4. 更新样本权重分布 ω

问题：怎样使用样本权重分布 ω ？

- Resampling (e.g. Roulette wheel, random number of uniform distribution)
- Add sample weight in loss function ($\min \frac{1}{N} \sum_{i=1}^N \omega_t^{(i)} \mathcal{L}(y^{(i)}, h_t(x^{(i)}))$)

问题：为何弱分类器的系数 $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ ？ 这是通过一个优化过程的得到的，解释推导如下：

Theorem 2 (The Training Error Bounds of Final Classifier).

$$\frac{1}{N} \sum_{i=1}^N I(G(x^{(i)}) \neq y^{(i)}) \leq \frac{1}{N} \sum_{i=1}^N \exp(-y^{(i)} f(x^{(i)})) = \prod_{t=1}^T z_t$$

- n : Number of training samples;
- T : Number of weak hypotheses;
- $f(x)$: The final hypothesis, $f(x) = \sum_{t=1}^T \alpha_t h_t(x)$;
- $G(x)$: The final classifier $G(x) = \text{sign}(f(x)) = \text{sign}(\sum_{t=1}^T \alpha_t h_t(x))$.

Proof. 先后证明上述连续不等式左右两个部分，具体证明过程如下：

1. For left part:

- if $G(x^{(i)}) \neq y^{(i)}$:

$$\begin{aligned} I(G(x^{(i)}) \neq y^{(i)}) &= 1 \Rightarrow y^{(i)} f(x^{(i)}) < 0 \\ &\Rightarrow -y^{(i)} f(x^{(i)}) > 0 \\ &\Rightarrow \exp(y^{(i)} f(x^{(i)})) > 1 \end{aligned}$$

- if $G(x^{(i)}) = y^{(i)}$:

$$\begin{aligned} y^{(i)} f(x^{(i)}) &> 0 \Rightarrow -y^{(i)} f(x^{(i)}) < 0 \\ &\Rightarrow \exp(-y^{(i)} f(x^{(i)})) \in (0, 1) \end{aligned}$$

- Therefore:

$$\frac{1}{N} \sum_{i=1}^N I(G(x^{(i)}) \neq y^{(i)}) \leq \frac{1}{N} \sum_{i=1}^N \exp(-y^{(i)} f(x^{(i)}))$$

2. For right part:

$$\begin{aligned} &\frac{1}{N} \sum_{i=1}^N \exp(-y^{(i)} f(x^{(i)})) \\ &= \frac{1}{N} \sum_{i=1}^N \exp\left(-y^{(i)} \sum_{t=1}^T \alpha_t h_t(x^{(i)})\right) \\ &= \sum_{i=1}^N \omega_1^{(i)} \exp\left(\sum_{t=1}^T -\alpha_t y^{(i)} h_t(x^{(i)})\right) \quad (\omega_1^{(i)} = \frac{1}{n}) \\ &= \sum_{i=1}^N \omega_1^{(i)} \prod_{t=1}^T \exp(-\alpha_t y^{(i)} h_t(x^{(i)})) \\ &= Z_1 \sum_{i=1}^N \frac{\omega_1^{(i)} \exp(-\alpha_1 y^{(i)} h_1(x^{(i)}))}{Z_1} \prod_{t=2}^T \exp(-\alpha_t y^{(i)} h_t(x^{(i)})) \\ &= Z_1 \sum_{i=1}^N \omega_2^{(i)} \prod_{t=2}^T \exp(-\alpha_t y^{(i)} h_t(x^{(i)})) \\ &= Z_1 Z_2 \sum_{i=1}^N \omega_3^{(i)} \prod_{t=3}^T \exp(-\alpha_t y^{(i)} h_t(x^{(i)})) \quad (\because \omega_{t+1}^{(i)} = \frac{\omega_t^{(i)} \exp(-\alpha_t y^{(i)} h_t(x^{(i)}))}{Z_t}) \\ &= Z_1 Z_2 \dots Z_{T-1} \sum_{i=1}^N \omega_T^{(i)} \exp(-\alpha_T y^{(i)} h_T(x^{(i)})) = \prod_{t=1}^T Z_t \end{aligned}$$

因此有：

$$\frac{1}{N} \sum_{i=1}^N I(G(x^{(i)}) \neq y^{(i)}) \leq \frac{1}{N} \sum_{i=1}^N \exp(-y^{(i)} f(x^{(i)})) = \prod_{t=1}^T z_t$$

而后优化上述不等式的上界：

$$\begin{aligned} & \min \prod_{t=1}^T Z_t \\ \Rightarrow & \min_{\alpha_t} Z_t \quad (\text{转换为独立优化任务}) \\ \Rightarrow & \min_{\alpha_t} \sum_{i=1}^N \omega_t^{(i)} \exp(-\alpha_t y^{(i)} h_t(x^{(i)})) \end{aligned}$$

因为是凸函数，所以最优解在梯度为 0 处，即：

$$\begin{aligned} & \frac{\partial \sum_{i=1}^N \omega_t^{(i)} \exp(-\alpha_t y^{(i)} h_t(x^{(i)}))}{\partial \alpha_t} = 0 \\ \Rightarrow & \sum_{i=1}^N \omega_t^{(i)} (-y^{(i)} h_t(x^{(i)})) \exp(-\alpha_t y^{(i)} h_t(x^{(i)})) = 0 \\ \Rightarrow & \alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right) \end{aligned}$$

13.2.2 Adaboost for multi classification

Algorithm 14: Adaboost.M1 for Multiclass Classification[?]

Input: Training dataset $\{\mathcal{X}, \mathcal{Y}\}$, where $\mathcal{X} \in \mathbb{R}^{n \times d}$, $\mathcal{Y} \in \mathbb{R}^n$, and $x^{(i)} \in \mathcal{X}$, $y^{(i)} \in \mathcal{Y} = \{1, 2, \dots, K\}$;

Weak hypothesis $h(x)$;

Number of weak hypothesis T.

Output: The final hypothesis $G(x) = \arg \max_{y \in \mathcal{Y}} \sum_{t: h_t(x) = y} \log \frac{1}{\beta_t}$.

1 Initialize samples weight distribution: $\omega_i^1 = \frac{1}{n}$, for all $i \in n$

2 **for** $t = 1, \dots, T$ **do**

3 Train the weak classifier $h_t(x)$, providing it with the distribution ω^t

4 Get weak hypothesis $h_t(x) : \mathcal{X} \rightarrow \mathcal{Y}$ with error:

$$\epsilon_t = \Pr_{\omega^t}(h_t(x^{(i)}) \neq y^{(i)}) = \sum_{i: h_t(x^{(i)}) \neq y^{(i)}} \omega_i^t$$

5 If $\epsilon_t > \frac{1}{2}$, then set $T = t - 1$ and abort loop

6 Choose $\beta_t = \frac{\epsilon_t}{1 - \epsilon_t} \in (0, 1)$

7 Update distribution ω^t :

$$\omega_{t+1}^{(i)} = \begin{cases} \frac{\omega_i^t}{Z_t} \times \beta_t & \text{if } h_t(x^{(i)}) = y^{(i)} \\ \frac{\omega_i^t}{Z_t} \times 1 & \text{if } h_t(x^{(i)}) \neq y^{(i)} \end{cases}$$

8 **end**

Adaboost.M1 的解释说明

- Adaboost.M1 算法中，要求每一个基分类器在多分类上的 0-1 损失都小于 $\frac{1}{2}$ （实际上这个条件有些不太好）；
- 当 $0 < \epsilon_t < \frac{1}{2}$ 时， ϵ_t 越小， $\beta_t \in (0, 1)$ 越小， $\frac{1}{\beta_t} \in (1, +\infty)$ 越大，那么 $\log \frac{1}{\beta_t} \in (1, +\infty)$ 越大，那么在最终分类器 $G(x)$ 中，该分类正确的弱分类器“话语权”更大；
- 因为 $\beta_t < 1$ ，所以正确分类的样本权重相对错误分类的样本而减小。

13.2.3 Adaboost for regression

Algorithm 15: Adaboost.R[?]

Input: Training dataset $\{\mathcal{X}, \mathcal{Y}\}$, where $\mathcal{X} \in \mathbb{R}^{n \times d}$, $\mathcal{Y} \in \mathbb{R}^n$, and $x^{(i)} \in \mathcal{X}$, $y^{(i)} \in \mathcal{Y}$;

Weak hypothesis $h(x)$;

Number of weak hypothesis T.

Output: The final hypothesis $G(x) = \sum_{t=1}^T \ln(\frac{1}{\alpha_t}) h_t(x)$.

1 Initialize samples weight distribution: $\omega_i^1 = \frac{1}{n}$, for all $i \in n$

2 **for** $t = 1, \dots, T$ **do**

3 Train the weak regressor $h_t(x)$, providing it with the distribution ω^t

4 Calculate the max error of all training samples:

$$E_t = \max |y^{(i)} - h_t(x^{(i)})|, i \in \{1, \dots, n\}$$

5 Calculate the relative errors $e_t(i)$ over all training samples: $e_t(i) = \frac{(y^{(i)} - h_t(x^{(i)}))^2}{E_t^2}$ (for square error), or $e_t(i) = \frac{|y^{(i)} - h_t(x^{(i)})|}{E_t}$ (for absolute error)

6 Calculate the error rate of regressor $h_t(x)$: $\epsilon_t = \sum_{i=1}^N \omega_i^t e_t(i)$

7 Calculate the weight of weak hypothesis $h_t(x)$: $\alpha_t = \frac{\epsilon_t}{1-\epsilon_t}$

8 Update distribution ω^t : $\omega_i^{t+1} = \frac{\omega_i^t \alpha_t^{1-\epsilon_t(i)}}{Z_t}$, $Z_t = \sum_{i=1}^m \omega_i^t \alpha_t^{1-\epsilon_t(i)}$

9 **end**

13.3 Gradient Boosting Machine

13.3.1 Model formulation

Algorithm 16: Gradient Boosting Machine Algorithm[?]

Input: Training dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^N$;
Number of iterations M ;
A differentiable loss function $\mathcal{L}(y, F(\mathbf{x}))$.

Output: The final hypothesis $F_M(x)$.

1 Initialize model with a constant value:

$$F_0(\mathbf{x}) = \arg \min_{\alpha} \sum_{i=1}^N \mathcal{L}(y^{(i)}, \alpha)$$

2 **for** $m = 1, \dots, M$ **do**

3 Compute so-called pseudo-residuals:

$$r_m^{(i)} = -\left[\frac{\partial \mathcal{L}(y^{(i)}, F(x^{(i)}))}{\partial F(x^{(i)})}\right]_{F(x)=F_{m-1}(x)}, i = 1, \dots, n$$

4 Fit a base learner (e.g. tree, lr) $h_m(x)$ to pseudo-residuals, namely train it using the training set:

$$\mathcal{D} = \{(x^{(i)}, r_m^{(i)})\}_{i=1}^N$$

5 Compute multiplier α_m by solving the following one-dimensional optimization problem:

$$\alpha_m = \arg \min_{\alpha} \sum_{i=1}^N \mathcal{L}(y^{(i)}, F_{m-1}(x^{(i)}) + \alpha h_m(x^{(i)}))$$

6 Update the model:

$$F_m(\mathbf{x}) := F_{m-1}(\mathbf{x}) + \alpha_m \cdot h_m(x)$$

7 **end**

可见在 GBM 的学习中，我们先初始化模型为一个常数，然后进入迭代过程。在每一步迭代中，我们需要做四件事情 [?] :

1. 计算出当前每个训练样本的残差 $r_m^{(i)}$;
2. 运用残差作为 label 训练一个弱学习器 $h_m(\mathbf{x})$;
3. 计算一个一维优化问题（线搜索）得到当前弱学习器的乘子系数 α_m ;
4. 更新模型 $F_m(\mathbf{x})$.

13.3.2 GBM regression

任意训练样本 $(\mathbf{x}^{(i)}, y^{(i)})$ 的 so-called pseudo-residuals :

$$\therefore \frac{\partial \mathcal{L}(y^{(i)}, F(\mathbf{x}^{(i)}))}{\partial F(\mathbf{x}^{(i)})} = \frac{\partial}{\partial F(\mathbf{x}^{(i)})} \frac{1}{2} (y^{(i)} - F(\mathbf{x}^{(i)}))^2$$

$$= - \left(y^{(i)} - F(\mathbf{x}^{(i)}) \right)$$

$$\begin{aligned}\therefore r_m^{(i)} &= - \left[\frac{\partial \mathcal{L}(y^{(i)}, F(\mathbf{x}^{(i)}))}{\partial F(\mathbf{x}^{(i)})} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \\ &= y^{(i)} - F_{m-1}(\mathbf{x}^{(i)}) \quad (\text{数值残差})\end{aligned}$$

13.3.3 GBM classification

因为分类问题的损失函数（针对单一样本 (\mathbf{x}, \mathbf{y}) ）为 Cross Entropy，即：

$$\mathcal{L}(\mathbf{y}, F(\mathbf{x})) = -\mathbf{y} \cdot \log F(\mathbf{x}) = - \sum_{k=1}^K y_k \log F(x_k)$$

因此任意训练样本 $(\mathbf{x}^{(i)}, y^{(i)})$ 的 so-called pseudo-residuals：

$$r_m^{(i)} = - \left[\frac{\partial \mathcal{L}(y^{(i)}, F(\mathbf{x}^{(i)}))}{\partial F(\mathbf{x}^{(i)})} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} = y^{(i)} - F_{m-1}(\mathbf{x}^{(i)}) \quad (\text{概率残差})$$

13.4 XGBoost

13.4.1 Model prediction

XGBoost[?] 模型的预测函数为：

$$F(\mathbf{x}) = \sum_{t=1}^T f_t(\mathbf{x})$$

13.4.2 Model training

Dataset 训练数据的表达如下：

- 训练数据集： $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$
- 特征向量： $\mathbf{x}^{(i)} \in \mathbb{R}^d$
- 样本标签： $y^{(i)}$

Model details 我们做如下符号表示：

- $f_t(\mathbf{x})$: 第 t 轮迭代训练所得的弱学习器
- \hat{y}_t : 第 t 轮训练结束后所得的强学习器，即：

$$\hat{y}_t = \sum_{h=1}^t f_h(\mathbf{x})$$

第 t 轮结束后需要学习到的强学习器（预测函数）可表示为：

$$\begin{aligned}\hat{y}_t &= \hat{y}_{t-1} + f_t(\mathbf{x}) \\ &= \sum_{h=1}^{t-1} f_h(\mathbf{x}) + f_t(\mathbf{x})\end{aligned}$$

Loss function 一般机器学习模型目标函数的表示结构如下：

$$Obj(\Theta) = \mathcal{L}(\Theta) + \Omega(\Theta)$$

其中：

- $\mathcal{L}(\Theta)$: 损失函数，衡量模型拟合数据的能力
- $\Omega(\Theta)$: 正则化项 (Regularization)，防止模型过拟合 (overfitting)

采取此结构，Xgboost 在第 t 轮迭代的学习目标为：

$$\begin{aligned}&\min \sum_{i=1}^N \mathcal{L}(y^{(i)}, \hat{y}_t^{(i)}) + \Omega(\hat{y}_t) \\ \Rightarrow &\min \sum_{i=1}^N \mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)} + f_t(\mathbf{x}^{(i)})\right) + \Omega\left(\sum_{h=1}^t f_h(\mathbf{x})\right)\end{aligned}$$

$$\begin{aligned}
&\Rightarrow \min \sum_{i=1}^N \mathcal{L}\left(y^{(i)}, \underbrace{\hat{y}_{t-1}^{(i)}}_{\text{已知常数}} + f_t(\mathbf{x}^{(i)})\right) + \Omega\left(\underbrace{\sum_{h=1}^{t-1} f_h(\mathbf{x})}_{\text{已知常数}} + f_t(\mathbf{x})\right) \\
&\Rightarrow \min \sum_{i=1}^N \mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)} + f_t(\mathbf{x}^{(i)})\right) + \Omega(f_t(\mathbf{x})) + \text{constant} \\
&\Rightarrow \min \sum_{i=1}^N \mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)} + f_t(\mathbf{x}^{(i)})\right) + \Omega(f_t(\mathbf{x}))
\end{aligned}$$

根据泰勒展开公式 (泰勒二阶近似展开) :

$$\begin{aligned}
\because f(x + \Delta x) &\simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2 \\
\therefore \mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)} + f_t(\mathbf{x}^{(i)})\right) &\simeq \mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)}\right) + \frac{\partial \mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)}\right)}{\partial \hat{y}_{t-1}^{(i)}} f_t(\mathbf{x}^{(i)}) + \frac{1}{2} \frac{\partial \mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)}\right)}{\partial^2 \hat{y}_{t-1}^{(i)}} f_t^2(\mathbf{x}^{(i)})
\end{aligned}$$

注意: 因为 $\mathcal{L}\left(y^{(i)}, \underbrace{\hat{y}_{t-1}^{(i)}}_{\text{对应 } x} + f_t(\mathbf{x}^{(i)})\right)$, 所以 $f'(x)$ 对应 $\frac{\partial \mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)}\right)}{\partial \hat{y}_{t-1}^{(i)}}$, 而不是 $\frac{\partial \mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)}\right)}{\partial f_t(\mathbf{x}^{(i)})}$

如果不好理解, 考虑回归问题的平方误差损失函数, 那么:

$$\begin{aligned}
\frac{\partial \mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)}\right)}{\partial \hat{y}_{t-1}^{(i)}} &= \frac{\partial}{\partial \hat{y}_{t-1}^{(i)}} \frac{1}{2} (y^{(i)} - \hat{y}_{t-1}^{(i)})^2 \\
&= -(y^{(i)} - \hat{y}_{t-1}^{(i)}) \\
\frac{\partial \mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)}\right)}{\partial^2 \hat{y}_{t-1}^{(i)}} &= -1
\end{aligned}$$

代入目标函数继续化简:

$$\begin{aligned}
&\min \sum_{i=1}^N \mathcal{L}\left(y^{(i)}, \hat{y}_t^{(i)}\right) + \Omega(\hat{y}_t) \\
&\Rightarrow \min \sum_{i=1}^N \underbrace{\mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)} + f_t(\mathbf{x}^{(i)})\right)}_{\text{利用泰勒展开等价替换}} + \Omega(f_t) \\
&\Rightarrow \min \sum_{i=1}^N \left(\underbrace{\mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)}\right)}_{\text{记作 } g_{t-1}^{(i)}} + \underbrace{\frac{\partial \mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)}\right)}{\partial \hat{y}_{t-1}^{(i)}} f_t(\mathbf{x}^{(i)})}_{\text{记作 } h_{t-1}^{(i)}} + \frac{1}{2} \underbrace{\frac{\partial \mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)}\right)}{\partial^2 \hat{y}_{t-1}^{(i)}} f_t^2(\mathbf{x}^{(i)})}_{\text{常数}} \right) + \Omega(f_t) \\
&\Rightarrow \min \sum_{i=1}^N \left(\underbrace{\mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)}\right)}_{\text{常数}} + g_{t-1}^{(i)} f_t(\mathbf{x}^{(i)}) + \frac{1}{2} h_{t-1}^{(i)} f_t^2(\mathbf{x}^{(i)}) \right) + \Omega(f_t) \\
&\Rightarrow \min \sum_{i=1}^N \left(g_{t-1}^{(i)} f_t(\mathbf{x}^{(i)}) + \frac{1}{2} h_{t-1}^{(i)} f_t^2(\mathbf{x}^{(i)}) \right) + \Omega(f_t)
\end{aligned}$$

问题: 为什么要这样费劲地用泰勒展开式写出来?

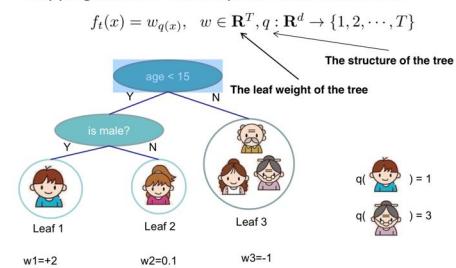
- 理论角度**: 把 $\mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)} + f_t(\mathbf{x}^{(i)})\right)$ 用泰勒展开写出来我们就明明白白地知道自己要学什么了，怎么收敛。
- 工程角度**: 目标函数的优化(学习)只取决于 $g^{(i)}$ 和 $h^{(i)}$ 所在的项，如果现在要求损失函数不是 squared loss 了，而是 logistic loss，那么这样把 $g^{(i)}$ 和 $h^{(i)}$ 显示地写出来就很有利于模型的便捷实现。

注意: XGBoost 使用 Taylor 二阶展开，一般的 GBM 只使用一阶展开

然而新的问题又来了

Refine the definition of tree

- We define tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf



Define Complexity of a Tree (cont')

- Define complexity as (this is not the only possible definition)

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

Number of leaves L2 norm of leaf scores

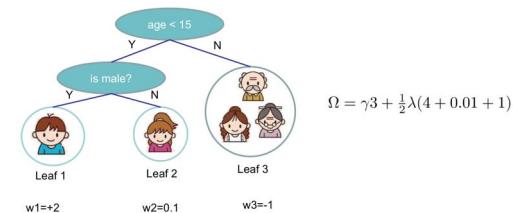


图 47: Refine the definition of tree

图 48: Define complexity of tree

- 如何表示 $f_t(\mathbf{x})$? 假定**树的结构已确定**, 那么有

$$f_t(\mathbf{x}) = w_{q(\mathbf{x})}$$

符号解释如下:

- T : 叶子结点个数
- $\mathbf{w} \in \mathbb{R}^T$
- $q: \mathbb{R}^d \rightarrow \{1, \dots, T\}$

步骤解释如下:

- \mathbf{x} 根据函数 $q(\cdot)$ 找到对应的叶子节点 $q(\mathbf{x})$
- 函数 f_t 返回叶子结点 $q(\mathbf{x})$ 上的估计值

- 如何定义 $\Omega(f_t)$?

$$\Omega(f_t(\mathbf{x})) = \gamma T + \frac{\lambda}{2} \sum_{j=1}^T w_j^2$$

其中:

- γT : 控制叶子结点个数
- $\frac{\lambda}{2} \sum_{j=1}^T w_j^2$: 控制叶子结点权重的大小

该定义陈天奇给出，所以并不是唯一可行的定义。

代入目标函数继续化简：

$$\begin{aligned}
 & \min \sum_{i=1}^N \mathcal{L}(y^{(i)}, \hat{y}_t^{(i)}) + \Omega(\hat{y}_t) \\
 \Rightarrow & \min \sum_{i=1}^N \left(g_{t-1}^{(i)} f_t(\mathbf{x}^{(i)}) + \frac{1}{2} h_{t-1}^{(i)} f_t^2(\mathbf{x}^{(i)}) \right) + \Omega(f_t) \\
 \Rightarrow & \min \sum_{i=1}^N \underbrace{\left(g_{t-1}^{(i)} w_{q(\mathbf{x}^{(i)})} + \frac{1}{2} h_{t-1}^{(i)} w_{q(\mathbf{x}^{(i)})}^2 \right)}_{\text{又不知道怎么化简处理了}} + \gamma T + \frac{\lambda}{2} \sum_{j=1}^T w_j^2
 \end{aligned}$$

因为 n 个训练样本都要落在叶子结点上，那么可以将上述损失函数表示为各个叶子结点损失函数之和，即第 j 个叶子结点的训练样本集合为：

$$I_j = \{i \mid q(\mathbf{x}^{(i)}) = j\}$$

代入上式中可得：

$$\begin{aligned}
 & \min \sum_{i=1}^N \mathcal{L}(y^{(i)}, \hat{y}_t^{(i)}) + \Omega(\hat{y}_t) \\
 \Rightarrow & \min \sum_{i=1}^N \left(g_{t-1}^{(i)} w_{q(\mathbf{x}^{(i)})} + \frac{1}{2} h_{t-1}^{(i)} w_{q(\mathbf{x}^{(i)})}^2 \right) + \gamma T + \frac{\lambda}{2} \sum_{j=1}^T w_j^2 \\
 \Rightarrow & \min \sum_{j=1}^T \sum_{i \in I_j} \left(g_{t-1}^{(i)} w_j + \frac{1}{2} h_{t-1}^{(i)} w_j^2 \right) + \gamma T + \frac{\lambda}{2} \sum_{j=1}^T w_j^2 \quad (\text{最关键步骤}) \\
 \Rightarrow & \min \sum_{j=1}^T \left(\sum_{i \in I_j} g_{t-1}^{(i)} w_j + \frac{1}{2} \sum_{i \in I_j} h_{t-1}^{(i)} w_j^2 \right) + \gamma T + \frac{\lambda}{2} \sum_{j=1}^T w_j^2 \\
 \Rightarrow & \min \sum_{j=1}^T \left(w_j \sum_{i \in I_j} g_{t-1}^{(i)} + \frac{1}{2} w_j^2 \left(\sum_{i \in I_j} h_{t-1}^{(i)} + \lambda \right) \right) + \gamma T
 \end{aligned}$$

那么之后的任务就是独立地优化 T 个二次函数。记：

$$\begin{aligned}
 G_{t-1}^{(j)} &= \sum_{i \in I_j} g_{t-1}^{(i)} \\
 H_{t-1}^{(j)} &= \sum_{i \in I_j} h_{t-1}^{(i)}
 \end{aligned}$$

所以上述目标函数又可以化简为：

$$\begin{aligned}
 & \min \sum_{i=1}^N \mathcal{L}(y^{(i)}, \hat{y}_t^{(i)}) + \Omega(\hat{y}_t) \\
 \Rightarrow & \min \sum_{j=1}^T \left(w_j \sum_{i \in I_j} g_{t-1}^{(i)} + \frac{1}{2} w_j^2 \left(\sum_{i \in I_j} h_{t-1}^{(i)} + \lambda \right) \right) + \gamma T \\
 \Rightarrow & \min \sum_{j=1}^T \left(w_j G_{t-1}^{(j)} + \frac{1}{2} w_j^2 (H_{t-1}^{(j)} + \lambda) \right) + \gamma T
 \end{aligned}$$

上述式子即为 Xgboost 模型最终的目标函数（二次凸函数）。

Optimization 因为二次凸函数有：

$$\begin{aligned}\arg \min_x Gx + \frac{1}{2}Hx^2 &= -\frac{G}{H} \\ \min_x Gx + \frac{1}{2}Hx^2 &= -\frac{1}{2}\frac{G^2}{H}\end{aligned}$$

所以可得目标函数的最优解及最小值分别为：

$$\begin{aligned}w_j^* &= -\frac{G_{t-1}^{(j)}}{H_{t-1}^{(j)} + \lambda} = -\frac{\sum_{i \in I_j} g_{t-1}^{(i)}}{\sum_{i \in I_j} h_{t-1}^{(i)} + \lambda} \\ obj &= -\frac{1}{2} \sum_{j=1}^T \frac{(G_{t-1}^{(j)})^2}{H_{t-1}^{(j)} + \lambda} + \gamma T\end{aligned}$$

注意如上的推导是基于树的结构（树的深度，叶子结点个数）确定的情况

问题：如何找到最好的划分结构？不可能遍历无穷种树结构，因此采取贪心选择，即：

1. 从深度为 0 的二叉树开始（一个根节点）
2. 对每个叶子节点尝试添加一个分割（split），将原来的数据分为左右两部分然后计算这样添加后目标函数的变化

问题：分割点划分的依据是什么？

$$\begin{aligned}Gain &= obj_{\text{before-split}} - obj_{\text{after-split}} \\ &= \left(-\frac{1}{2} \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} + \gamma \right) - \left(-\frac{1}{2} \frac{G_L^2}{H_L + \lambda} + \gamma \right) - \left(-\frac{1}{2} \frac{G_R^2}{H_R + \lambda} + \gamma \right) \\ &= -\frac{1}{2} \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} + \gamma + \frac{1}{2} \frac{G_L^2}{H_L + \lambda} - \gamma + \frac{1}{2} \frac{G_R^2}{H_R + \lambda} - \gamma \\ &= \frac{1}{2} \left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right) - \gamma\end{aligned}$$

实际上在找一个最优分割点时，我们全部要做的，就是计算每一边儿样本的 g_i 和 h_i 之和，然后计算一下 Gain。

问题：如何找到最优分割点？遍历所有特征

- numerical feature：排序所有特征值，然后以任意大小相邻的两个特征值的中点作为分割点，计算 Gain.
- categorical feature：(one-hot 后) 按照当前 one-hot 离散列是否为 1 分成两部分，计算 Gain.

精确找寻切分点算法如下：

Algorithm 17: Basic Exact Greedy Algorithm for Split Finding

Input: Instances set of current node: $\mathcal{I}^{m \times d}$.
Output: Split with max score.

```

score ← −inf
 $G \leftarrow \sum_{i \in \mathcal{I}} g^{(i)}$ ,  $H \leftarrow \sum_{i \in \mathcal{I}} h^{(i)}$ 
for  $j = 1, \dots, d$  do
     $G_L \leftarrow 0$ ,  $H_L \leftarrow 0$ 
    for  $i$  in sorted( $\mathcal{I}$ , by  $x_j^{(i)}$ ) do
         $G_L \leftarrow G_L + g^{(i)}$ ,  $H_L \leftarrow H_L + h^{(i)}$ 
         $G_R \leftarrow G - G_L$ ,  $H_R \leftarrow H - H_L$ 
         $score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$ 
    end
end
```

近似找寻切分点算法如下（先对特征进行百分比分箱）：

Algorithm 18: Approximate Algorithm for Split Finding

```

for  $j = 1, \dots, d$  do
    Propose  $S_j = \{s_{j1}, s_{j2}, \dots, s_{jl}\}$  by percentiles on feature  $j$ 
    Proposal can be done per tree (global), or per split (local).
end
for  $j = 1, \dots, d$  do
     $G_{jv} \leftarrow \sum_{j \in \{j \mid s_{j,v} \geq x_j^{(i)} \geq s_{j,v-1}\}} g^{(i)}$ 
     $H_{jv} \leftarrow \sum_{j \in \{j \mid s_{j,v} \geq x_j^{(i)} \geq s_{j,v-1}\}} h^{(i)}$ 
end
```

Follow same step as in previous section to find max score only among proposed splits.

13.4.3 Model pruning

全都围绕最优分割是否是 negative gain

- Pre-stopping: Stop split if the best split have negative gain, but maybe a split can benefit future splits..
- Post-Pruning: Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain.

13.5 LightGBM

13.5.1 Introduction

论文出处：[LightGBM: A Highly Efficient Gradient Boosting Decision Tree\[?\]](#)。

GBDT 性能问题 Gradient Boosting Decision Tree (GBDT) 是一个很流行的机器学习算法理论框架，也有很多优秀的开源实现 (e.g. XGBoost, pGBDT)。然而虽然有很多工程上的优化方法已经被添加进了 GBDT 的工程实现中，但是当数据量很大，数据维度很高时，算法框架的有效性和可扩展性依然得不到满足。

原因分析 训练阶段，在每一棵 CART，每一个最优分割点的训练寻找过程中，需要计算所有候选分割值（特征 + 特征值）的损失，然后找到最小损失值对应的分割点作为最优分割点。因此时间的花费取决于：

- 训练样本的个数 N
- 训练样本的维度 d

太费时间了，还是一棵树就如此！

解决办法

- 减少训练数据的样本个数 N (number of instances)
- 减少训练数据的特征个数 d (number of features)

但是该办法不可行。因为不知道如何进行采样。因为这是 Gradient Boosting Machine，不是 Adaboost，这里面没有样本权重 (sample weight)。因此引出了 LightGBM 模型，该模型实际上包含两个算法：

- Gradient-based One-Side Sampling (GOSS): 减少训练样本个数 N
- Exclusive Feature Bundling (EFB): 减少训练特征维度 d

13.5.2 Model training

Algorithm 19: Gradient-based One-Side Sampling Algorithm[?]

Input: Training dataset $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$, where $\mathbf{x}^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$;

Number of base tree model (CART): T;

Sampling ratio of large gradient data: α ;

Sampling ratio of small gradient data: β ;

Loss function: $loss$;

Weak learner: $f(\mathbf{x})$.

Output: The final hypothesis $F(\mathbf{x}) = \sum_{t=1}^T f_t(\mathbf{x})$.

```

1 Models ← {}, fact ←  $\frac{1-\alpha}{\beta}$ 
2 Compute the number of samples to keep (with large gradients): topN =  $\alpha \times n$ 
3 Compute the number of samples to be random sampling (with small gradients): randN =  $\beta \times n$ 
4 for  $t = 1$  to  $T$  do
5    $\mathbf{preds} \in \mathbb{R}^n \leftarrow \underbrace{\text{models.predict}(\mathcal{X}^{N \times d})}_{\text{先前的模型进行预测}}$ 
6    $\underbrace{\text{Compute the gradients of current Models over all samples: } \mathbf{g} \leftarrow loss(\mathbf{y}, \mathbf{preds})}_{\text{注意是计算所有原始样本的残差}}$ 
7   Initialize relative weights of all samples:  $\mathbf{w} = \underbrace{\{1, 1, \dots, 1\}}_{\text{注意是计算所有原始样本}} \underbrace{\text{length: N}}$ 
8   Sort all samples with indices based on calculated abs of gradients:
       $\underbrace{\text{根据梯度的绝对值排序所有样本，排序记录样本索引（省空间）}}_{\text{sorted}} \leftarrow \text{GetSortedIndices}(\text{abs}(\mathbf{g}))$ 
       $\underbrace{\text{topSet} \leftarrow \text{sorted}[1 : \text{topN}]}_{\text{前 topN 个大梯度（残差）的样本全部保留}}$ 
9    $\underbrace{\text{randSet} \leftarrow \text{RandomPick}(\text{sorted}[\text{topN} : N], \text{randN})}_{\text{从除去梯度最大的 topN 个样本的样本中随机抽取 randN 个样本，保留下采样}}$ 
10   $\underbrace{\text{usedSet} \leftarrow \text{topSet} + \text{randSet}}_{\text{Models 预测后得到的残差}}$ 
11  Assign weight fact to the small gradient data:  $\mathbf{w}[\text{randSet}] \times = fact$ 
12  Train a weak learner  $f_t(\mathbf{x})$ , using dataset  $\{ \mathcal{X}[\text{usedSet}], \underbrace{-\mathbf{g}[\text{usedSet}]}_{\text{Models 预测后得到的残差}} \}$  with sample
13  weight distribution  $\mathbf{w}[\text{randSet}]$ .
14  Models.append( $f_t(\mathbf{x})$ )
15 end

```

13.5.3 Reference

- GPU-acceleration for Large-scale Tree Boosting[?]
- A Communication-Efficient Parallel Algorithm for Decision Tree[?]

13.6 CatBoost (未完成)

13.6.1 Introduction

论文出处：[CatBoost: unbiased boosting with categorical features\[?\]](#) 和[CatBoost: gradient boosting with categorical features support\[?\]](#)

13.6.2 Model formulation

Categorical feature

13.6.3 Reference

- Yandex CatBoost: a machine learning method based on gradient boosting over decision trees
- Datacruiser's Blog: CatBoost 算法疏理

13.7 NGBoost (未完成)

13.7.1 Introduction

根据论文NGBoost: Natural Gradient Boosting for Probabilistic Prediction[?]

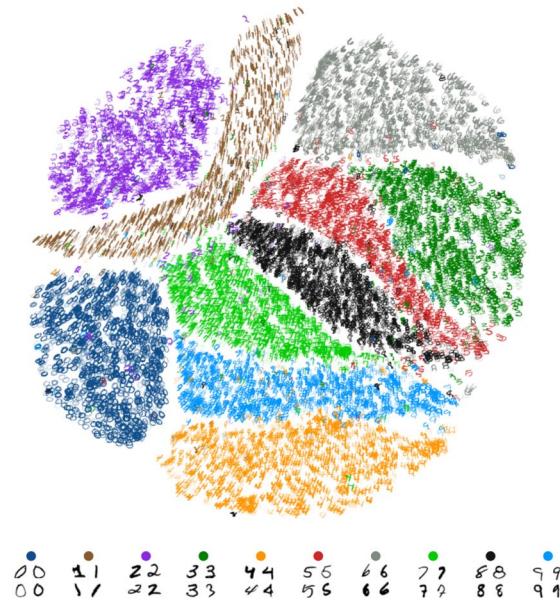
14 Dimensionality Reduction

14.1 t-Distributed Stochastic Neighbor Embedding (t-SNE)

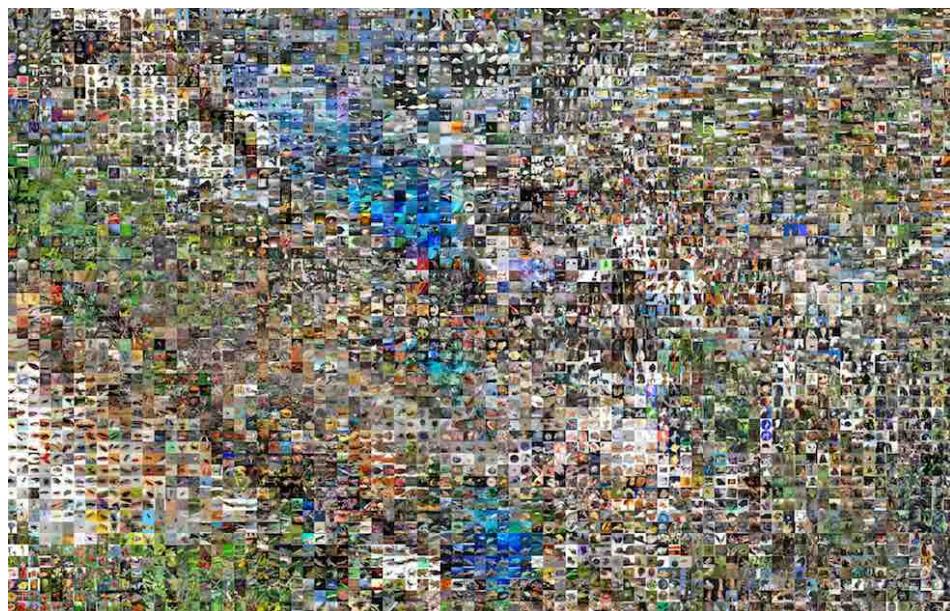
14.1.1 Introduction

论文出处：[Visualizing Data using t-SNE\[?\]](#) 以及 [Accelerating t-SNE using Tree-Based Algorithms\[?\]](#)。
t-SNE 本质上是一个需要通过优化得到最优参数的学习算法。一些应用如下所示：

- tSNE embedding of the MNIST dataset. Cite image from thesis: [Dimensionality-Reduction Algorithms for Progressive Visual Analytics](#).



- Gridded t-SNE of images of animals from [Caltech101](#).



14.1.2 Model formulation

Methodology t-SNE 算法的目的是：将样本集中的高维样本特征 $\mathcal{D}_X = \{x^{(i)} \in \mathbb{R}^{\mathcal{X}}\}_{i=1}^N$ 表示为对应的第维样本特征 $\mathcal{D}_Y = \{y^{(i)} \in \mathbb{R}^{\mathcal{Y}}\}_{i=1}^N$ ($\mathbb{R}^{\mathcal{Y}} \ll \mathbb{R}^{\mathcal{X}}$)，其算法过程包括三个部分：

- **高维空间距离转概率**。将高维空间内，样本集合 \mathcal{D}_X 中任意样本对 $x^{(i)}, x^{(j)} \in \mathbb{R}^{\mathcal{X}}$ ($i, j \in \{1, \dots, N\}$, $i \neq j$) 之间的**距离转换为条件概率** (i.g. affinities) :

$$p(x^{(j)}|x^{(i)}) = \frac{\exp\left(-\frac{\|x^{(j)} - x^{(i)}\|^2}{2\sigma_i^2}\right)}{\sum_{k=1, k \neq i}^N \exp\left(-\frac{\|x^{(k)} - x^{(i)}\|^2}{2\sigma_i^2}\right)}$$

其中 σ_i 为以样本点 $x^{(i)}$ 为中心的高斯分布 ($(x^{(k)} - x^{(i)}) | k \neq i$) 的方差 (variance)。然后通过条件概率公式得到样本点 $x^{(i)}$ 和 $x^{(j)}$ 的**联合概率** $p(x^{(i)}, x^{(j)})$:

$$\begin{aligned} p(x^{(i)}, x^{(j)}) &= p(x^{(i)}) \cdot p(x^{(j)}|x^{(i)}) \\ &= p(x^{(j)}) \cdot p(x^{(i)}|x^{(j)}) \end{aligned}$$

因此通过**加权**因子 $\alpha \in [0, 1]$ 综合上述两种先验概率的概率计算方式，通过样本集内的点计算得到点 $x^{(i)}$ 和 $x^{(j)}$ 的联合概率 $p(x^{(i)}, x^{(j)})$ 为：

$$\begin{aligned} p(x^{(i)}, x^{(j)}) &= \alpha \cdot p(x^{(i)}) \cdot p(x^{(j)}|x^{(i)}) + (1 - \alpha) \cdot p(x^{(j)}) \cdot p(x^{(i)}|x^{(j)}) \quad (\alpha \in [0, 1]) \\ &= \frac{\alpha}{N} \cdot p(x^{(j)}|x^{(i)}) + \frac{1 - \alpha}{N} \cdot p(x^{(i)}|x^{(j)}) \quad (\because i.i.d, \therefore p^{(i)} = \frac{1}{N}) \\ &= \frac{\alpha}{N} \cdot \underbrace{\frac{\exp\left(-\frac{\|x^{(j)} - x^{(i)}\|^2}{2\sigma_i^2}\right)}{\sum_{k=1, k \neq i}^N \exp\left(-\frac{\|x^{(k)} - x^{(i)}\|^2}{2\sigma_i^2}\right)}}_{p(x^{(j)}|x^{(i)})} + \frac{1 - \alpha}{N} \cdot \underbrace{\frac{\exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma_j^2}\right)}{\sum_{k=1, k \neq j}^N \exp\left(-\frac{\|x^{(k)} - x^{(j)}\|^2}{2\sigma_j^2}\right)}}_{p(x^{(i)}|x^{(j)})} \end{aligned}$$

若令 $\alpha = \frac{1}{2}$ ，则上述联合概率为：

$$p(x^{(i)}, x^{(j)}) = \frac{1}{2N} \cdot \left(\frac{\exp\left(-\frac{\|x^{(j)} - x^{(i)}\|^2}{2\sigma_i^2}\right)}{\sum_{k=1, k \neq i}^N \exp\left(-\frac{\|x^{(k)} - x^{(i)}\|^2}{2\sigma_i^2}\right)} + \frac{\exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma_j^2}\right)}{\sum_{k=1, k \neq j}^N \exp\left(-\frac{\|x^{(k)} - x^{(j)}\|^2}{2\sigma_j^2}\right)} \right)$$

Idea. 可以考虑将 α 作为一个**attention weight** 进行学习，做到：当 $p(x^{(j)}|x^{(i)})$ 相对于 $p(x^{(i)}|x^{(j)})$ 在 $p(x^{(i)}, x^{(j)})$ 需要提供更多的信息时， $\frac{1}{2} \ll \alpha \leq 1$ 。

因此最终得到原始高维空间 \mathcal{X} 内样本之间的联合概率分布 (如果将概率分布视为样本，则只是一条样本)：

$$\mathcal{P} = \{p(x^{(1)}, x^{(2)}), p(x^{(1)}, x^{(3)}), \dots, p(x^{(i)}, x^{(j)}), \dots, p(x^{(N)}, x^{(N-1)})\} \quad (i \neq j)$$

- **低维空间距离转概率**。其中低维空间 \mathcal{Y} 中任意向量 $y^{(i)} \in \mathbb{R}^{\mathcal{Y}}$ 是待学习优化的参数，即：

$$\theta = (y^{(1)}, \dots, y^{(i)}, \dots, y^{(N)}) \in \mathbb{R}^{N \times |\mathcal{Y}|}$$

其中 $y^{(i)} \in \mathbb{R}^{\mathcal{Y}}$ 为参数矩阵 $\theta \in \mathbb{R}^{N \times |\mathcal{Y}|}$ 中的第 i 行：

$$y^{(i)} = \theta_{((i), \cdot)}$$

而 $q(x^{(i)}, x^{(j)})$ 的计算和 $p(x^{(i)}, x^{(j)})$ 稍有不同，其直接根据距离计算联合概率分布而不是先由距离计算条件概率分布，再由条件概率分布计算联合概率分布。

$$q(x^{(i)}, x^{(j)}) = \frac{\frac{1}{1 + \|y^{(j)} - y^{(i)}\|^2}}{\sum_{i'=1}^N \sum_{j'=1, j' \neq i'}^N \frac{1}{1 + \|y^{(j')} - y^{(i')}\|^2}}$$

- **最小化联合概率分布差异**。最小化高维空间 \mathcal{X} 内样本联合概率分布 \mathcal{P} 和低维空间 \mathcal{Y} 内样本联合概率分布 \mathcal{Q} 之间的差异：

$$\begin{aligned} \min_{\theta} D_{\text{KL}}(\mathcal{P} \| \mathcal{Q}) &= \sum_{(i,j) \in \{1, \dots, N\}, i \neq j} p(x^{(i)}, x^{(j)}) \log \frac{p(x^{(i)}, x^{(j)})}{q(x^{(i)}, x^{(j)}; \theta)} \\ \Rightarrow \min_{\theta} D_{\text{KL}}(\mathcal{P} \| \mathcal{Q}) &= \sum_{i=1}^N \sum_{j=1, j \neq i}^N p(x^{(i)}, x^{(j)}) \log \frac{p(x^{(i)}, x^{(j)})}{q(x^{(i)}, x^{(j)}; \theta)} \end{aligned}$$

Objective

$$\min_{\theta} D_{\text{KL}}(\mathcal{P} \| \mathcal{Q}) = \sum_{i=1}^N \sum_{j=1, j \neq i}^N p(x^{(i)}, x^{(j)}) \log \frac{p(x^{(i)}, x^{(j)})}{q(x^{(i)}, x^{(j)}; \theta)}$$

其中：

- ground truth probability distribution 为：

$$\mathcal{P} = \left(p(x^{(i)}, x^{(j)}) \right)_{i,j \in \{1, \dots, N\}, i \neq j}$$

- predicted probability distribution 为：

$$\mathcal{Q} = \left(q(x^{(i)}, x^{(j)}; \theta) \right)_{i,j \in \{1, \dots, N\}, i \neq j}$$

Perplexity in t-SNE

- 算法中的困惑度 (perplexity) $\text{Perp}(\cdot)$ 是什么？

$$\begin{aligned} \text{Perp}(\mathcal{P}(x^{(i)})) &= 2^{\mathcal{H}(\mathcal{P}(x^{(i)}))} \\ &= 2^{-\sum_{j=1, j \neq i}^N p(x^{(j)} | x^{(i)}) \cdot \log p(x^{(j)} | x^{(i)})} \end{aligned}$$

而后求解方程：

$$\text{Perp}(\mathcal{P}(x^{(i)})) = Perp$$

即可得到 σ_i 。

- 为什么引入困惑度 (perplexity)？
- 如何选择？Automatic Selection of t-SNE Perplexity[?]
一般取值设置在 5~50 之间。

Training procedure 训练算法的伪代码如下：

Algorithm 20: Simple version of t-Distributed Stochastic Neighbor Embedding[?]

Input: Dataset $\mathcal{D}_X = \{x^{(i)} \in \mathbb{R}^{\mathcal{X}}\}_{i=1}^N$;

perplexity Attention weight of affinity α (Suggested default: $\frac{1}{2}$).

Output: The final parameters $\theta \in \mathbb{R}^{N \times |\mathcal{Y}|}$.

- 1 Compute original high-dimensional pairwise affinities $p(x^{(j)}|x^{(i)})$ with perplexity $Prep$ using equation:

$$p(x^{(j)}|x^{(i)}) = \frac{\exp\left(-\frac{\|x^{(j)} - x^{(i)}\|^2}{2\sigma_i^2}\right)}{\sum_{k=1, k \neq i}^N \exp\left(-\frac{\|x^{(k)} - x^{(i)}\|^2}{2\sigma_i^2}\right)}$$

- 2 Set original high-dimensional joint probability distribution $p(x^{(i)}, x^{(j)})$:

$$p(x^{(i)}, x^{(j)}) = \frac{\alpha}{N} \cdot p(x^{(j)}|x^{(i)}) + \frac{1-\alpha}{N} \cdot p(x^{(i)}|x^{(j)})$$

- 3 Initialize embeddings of each sample in lower dimension space \mathcal{Y} from $\mathcal{N}(0, 10^{-4}\mathcal{I})$:

$$\mathcal{D}_Y = \left\{y^{(i)} = \theta_{((i), \cdot)} \in \mathbb{R}^{\mathcal{Y}}\right\}_{i=1}^N$$

- 4 **while** stopping criterion not met **do**

- 5 Compute low-dimensional pairwise affinities $q(x^{(i)}, x^{(j)})$ using equation:

$$q(x^{(i)}, x^{(j)}) = \frac{1}{\sum_{i'=1}^N \sum_{j'=1, j' \neq i'}^N \frac{1}{1 + \|y^{(j')} - y^{(i')}\|^2}}$$

- 6 Update parameters θ by one step of gradient descent using:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \sum_{i=1}^N \sum_{j=1, j \neq i}^N p(x^{(i)}, x^{(j)}) \log \frac{p(x^{(i)}, x^{(j)})}{q(x^{(i)}, x^{(j)}; \theta)}$$

- 7 **end**
-

14.1.3 Reference

- Distill: How to Use t-SNE Effectively
- Dimensionality-Reduction Algorithms for Progressive Visual Analytics

Part III

Deep Learning Model and Theory

15 Dense Neural Networks (DNNs)

15.1 Fully Connection (FC)

15.1.1 Overview

Deep Learning[?] 的本质：

- 通过结构 cover 住数据特征之间的非线性
- 实现自动的特征抽取
- 达到端到端 (end-to-end) 的效果

15.1.2 Derivation of forward-propagation and back-propagation in fully-connected layer (未完成)

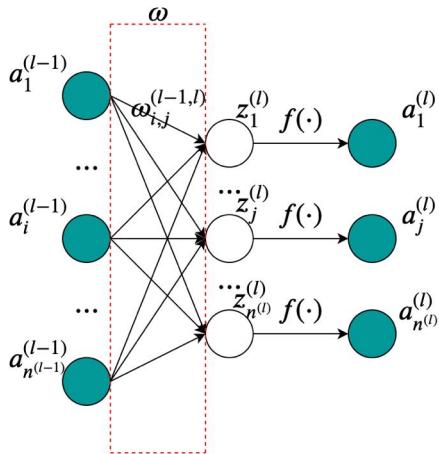


图 49：全链接层前向推导过程中神经元的关系

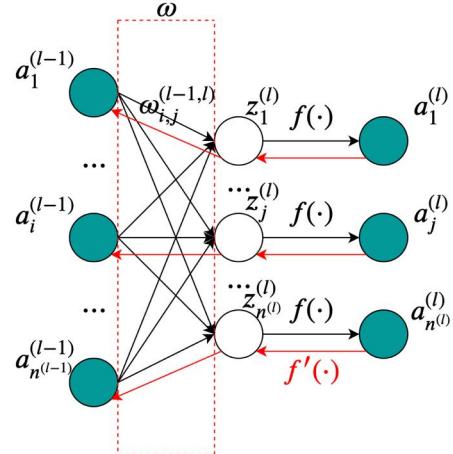


图 50：全链接层反向优化过程中神经元的关系

forward-propagation 分别从标量运算 (scalar operation) 和张量运算 (tensor operation) 的角度分析正向推导过程。

scalar operation

$$\begin{aligned} z_j^{(l)} &= \sum_{i=1}^{n^{(l-1)}} a_i^{(l-1)} \times w_{i,j}^{(l-1,l)} \\ a_j^{(l)} &= f(z_j^{(l)}) \end{aligned}$$

其中：

- $a_i^{(l-1)}, w_{i,j}^{(l-1,l)}, z_j^{(l)}, a_j^{(l)} \in \mathbb{R}$
- $f(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$

tensor operation

$$\begin{aligned} \mathbf{z}^{(l)} &= \mathbf{a}^{(l-1)} \cdot \mathbf{w}^{(l-1,l)} \\ \mathbf{a}^{(l)} &= f(\mathbf{z}^{(l)}) \quad (\text{element-wise operation}) \end{aligned}$$

其中：

- $\mathbf{a}^{(l-1)} \in \mathbb{R}^{N \times D^{(l-1)}}$
- $\mathbf{w}^{(l-1,l)} \in \mathbb{R}^{D^{(l-1)} \times D^{(l)}}$
- $\mathbf{z}^{(l)}, \mathbf{a}^{(l)} \in \mathbb{R}^{N \times D^{(l)}}$

back-propagation 分别从标量运算 (scalar operation) 和张量运算 (tensor operation) 的角度分析反向传播过程。注意：激活函数作用前后的导数为全导数，而没有偏导数。

scalar operation

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial z_j^{(l)}} &= \frac{\partial \mathcal{L}}{\partial a_j^{(l)}} \cdot \frac{da_j^{(l)}}{dz_j^{(l)}} \\ \frac{\partial \mathcal{L}}{\partial w_{i,j}^{(l-1,l)}} &= \frac{\partial \mathcal{L}}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{i,j}^{(l-1,l)}} \\ &= \frac{\partial \mathcal{L}}{\partial a_j^{(l)}} \cdot \frac{da_j^{(l)}}{dz_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{i,j}^{(l-1,l)}} \\ \frac{\partial \mathcal{L}}{\partial a_i^{(l-1)}} &= \sum_{j=1}^{n^{(l)}} \left(\frac{\partial \mathcal{L}}{\partial z_j^{(l)}} \cdot \frac{dz_j^{(l)}}{da_i^{(l-1)}} \right) \\ &= \sum_{j=1}^{n^{(l)}} \left(\frac{\partial \mathcal{L}}{\partial a_j^{(l)}} \cdot \frac{da_j^{(l)}}{dz_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial a_i^{(l-1)}} \right)\end{aligned}$$

tensor operation

$$\begin{aligned}\nabla_{\mathbf{z}^{(l)}} \mathcal{L} &= \nabla_{\mathbf{a}^{(l)}} \mathcal{L} \cdot \text{diag} \left(\frac{da^{(l)}}{dz^{(l)}} \right) \\ \nabla_{\mathbf{w}^{(l-1,l)}} \mathcal{L} &= \nabla_{\mathbf{z}^{(l)}} \mathcal{L} \cdot \nabla_{\mathbf{w}^{(l-1,l)}} \mathbf{z}^{(l)}\end{aligned}$$

其中将张量运算展开来看：

$$\begin{aligned}\nabla_{\mathbf{z}^{(l)}} \mathcal{L} &= \underbrace{\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_1^{(1;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial z_j^{(1;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial z_D^{(1;l)}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial \mathcal{L}}{\partial z_1^{(n;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial z_j^{(n;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial z_D^{(n;l)}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial z_1^{(N;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial z_j^{(N;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial z_D^{(N;l)}} \end{bmatrix}}_{\mathbb{R}^{N \times D^{(l)}}} & \quad (\text{上角标 } (n;l) \text{ 表示在神经网络第 } l \text{ 层针对第 } n \text{ 个样本}) \\ &= \underbrace{\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial a_1^{(1;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_j^{(1;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_D^{(1;l)}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial \mathcal{L}}{\partial a_1^{(n;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_j^{(n;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_D^{(n;l)}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial a_1^{(N;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_j^{(N;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_D^{(N;l)}} \end{bmatrix}}_{\mathbb{R}^{N \times D^{(l)}}} \cdot \underbrace{\begin{bmatrix} \frac{da_1^{(1;l)}}{dz_1^{(1;l)}} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{da_j^{(n;l)}}{dz_j^{(n;l)}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \frac{da_D^{(N;l)}}{dz_D^{(N;l)}} \end{bmatrix}}_{\text{diag} \left(\frac{da^{(l)}}{dz^{(l)}} \right) \in \mathbb{R}^{D^{(l)} \times D^{(l)}}} \\ &= \underbrace{\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial a_1^{(1;l)}} \frac{da_1^{(1;l)}}{dz_1^{(1;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_j^{(1;l)}} \frac{da_j^{(1;l)}}{dz_j^{(1;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_D^{(1;l)}} \frac{da_D^{(1;l)}}{dz_D^{(1;l)}} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial a_1^{(n;l)}} \frac{da_1^{(n;l)}}{dz_1^{(n;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_j^{(n;l)}} \frac{da_j^{(n;l)}}{dz_j^{(n;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_D^{(n;l)}} \frac{da_D^{(n;l)}}{dz_D^{(n;l)}} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial a_1^{(N;l)}} \frac{da_1^{(N;l)}}{dz_1^{(N;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_j^{(N;l)}} \frac{da_j^{(N;l)}}{dz_j^{(N;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_D^{(N;l)}} \frac{da_D^{(N;l)}}{dz_D^{(N;l)}} \end{bmatrix}}_{\mathbb{R}^{N \times D^{(l)}}}\end{aligned}$$

$$\nabla_{\mathbf{w}^{(l-1,l)}} \mathbf{z}^{(l)} = \left[\frac{\partial z^{(1;l)}}{\partial w_1^{(l-1,l)}} \right]$$

16 Convolutional Neural Networks (CNNs)

16.1 Convolution (未完成)

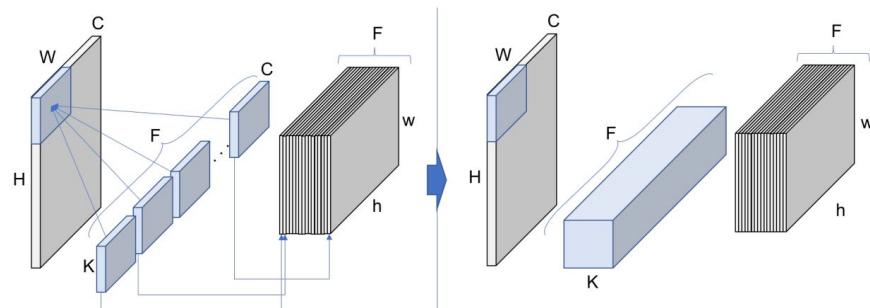


图 50: Different representation for convolutional kernels. Source: [pabloruizruiz10: Convolution operation](#)

16.1.1 Convolution as matrix multiplication

16.1.2 Derivation of forward-propagation and back-propagation in CNN (未完成, 重要)

Convolution layer

- Input feature map: $X \in \mathbb{R}^{W \times H \times C}$ or $\mathbb{R}^{C \times H \times W}$

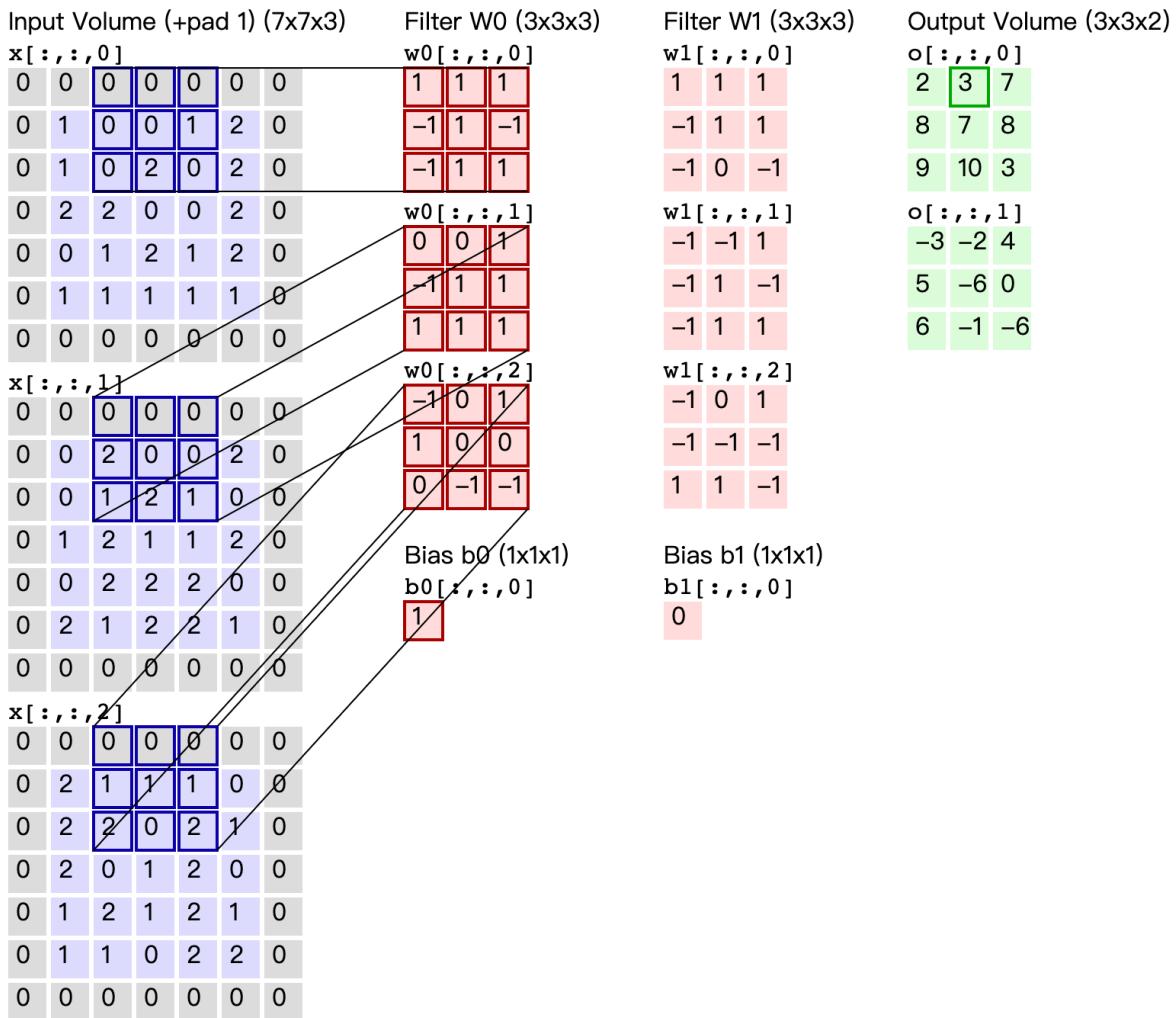


图 51: Convolution operation demo with $W_{\text{input}} = 5$, $H_{\text{input}} = 5$, $C_{\text{input}} = 3$, $K = 2$, $F = 3$, $S = 2$, $P = 1$.

16.1.3 Reference

- Derivation of Backpropagation in Convolutional Neural Network (CNN) [?]
- Deep learning for pedestrians: backpropagation in CNNs [?]
- Jake Bouvrie-MIT: Notes on Convolutional Neural Networks
- jefkine: Backpropagation In Convolutional Neural Networks
- CS231n: Convolutional Neural Networks for Visual Recognition
- denizyuret.github.io: Convolutional Neural Networks

16.1.4 Summary

- 卷机神经网络 (CNN) 最重要的作用就是抽取**局部特征**。
- 卷机神经网络架构的发展：
 1. LeNet5[?]: 1998
 2. AlexNet[?]: 2012
 3. VGG[?]: 2014
 4. GoogLeNet[?]: 2014 (2015.02 update to V3)
 5. ResNet[?]: 2015.12
 6. DenseNet[?]: 2016.08
 7. SqueezeNet[?]: 2016.11
 8. Xception[?]: 2016.07 (2017.04 update to V3)

16.2 Transposed Convolution (未完成)

16.2.1 Derivation of forward-propagation and back-propagation in Transposed Convolution (未完成, 重要)

forward-propagation

back-propagation

16.2.2 Reference

- A guide to convolution arithmetic for deep learning[?]
- Distill: Deconvolution and Checkerboard Artifacts[?]

16.3 LeNet5

16.3.1 Model formulation

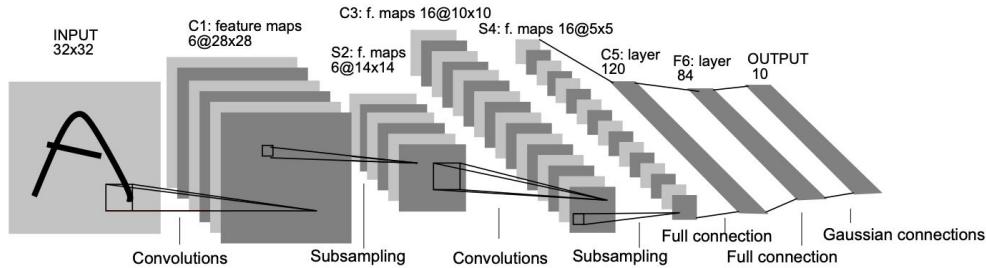


图 52: Architecture of LeNet5[?]

16.3.2 Model implement

MNIST[?] 手写体识别数据集（也可选择 Imagenet 数据集 [?]）存储方式如图，每行为一个样本，第一列为 label，第二列～最后一列为像素点

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1	0	0	0	0	0	0	0	0	9	8	0	0	34	29	7	0	11	24	0	0	3	3	1	0	1
2	1	0	0	0	0	0	0	0	0	0	0	0	209	190	181	150	170	193	180	219	5	0	0	0	0
3	2	0	0	0	0	0	0	0	14	53	99	17	0	0	0	0	0	0	0	12	94	68	14	0	0
4	2	0	0	0	0	0	0	0	0	0	0	0	161	212	138	150	169	164	176	202	255	183	26	0	0
5	3	0	0	0	0	0	0	0	0	0	0	0	0	37	0	0	0	0	0	0	17	0	0	0	0
6	2	0	0	0	0	0	0	0	0	44	105	44	10	0	0	0	0	0	0	0	0	0	34	684	34
7	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
8	6	0	0	0	0	0	0	0	0	0	0	0	0	108	25	0	0	0	132	54	0	0	0	2	0
9	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	0	0	0	0	92	66	0	0	0	81	91	0	0	0	0	0
11	3	0	0	0	0	0	0	0	0	1	0	0	0	83	142	50	50	0	0	85	145	31	0	0	0
12	4	0	0	0	0	0	0	0	0	1	1	0	0	2	153	100	88	81	138	50	0	0	0	0	0
13	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
14	6	0	0	0	0	0	0	0	0	1	1	0	0	0	142	122	94	95	138	177	13	0	0	2	1
15	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
16	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
17	6	0	0	0	0	0	0	0	0	1	5	0	0	0	91	167	142	143	165	148	0	0	0	1	1
18	3	0	0	0	0	0	0	0	0	0	15	132	121	148	148	145	151	154	138	120	145	0	0	0	
19	6	0	0	0	0	0	0	0	0	2	2	2	0	114	255	210	205	253	112	0	0	2	3	0	0
20	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	129	113	149	115	0	0	0	0	0
21	a	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	

图 53: The dataset saving in practical

```
# -*- coding: utf-8 -*-
"""
Created on 2018/12/28 15:39
author: Tong Jia
email: cecilio.jia@gmail.com
software: PyCharm
Description:
    An implement of LeNet5 using TensorFlow framework.
"""

import tensorflow as tf
from tensorflow.contrib.layers import l2_regularizer
from six.moves import xrange

def input_fn(
    filenames,
    delimiter,
    epochs,
    batch_size,
    shuffle=True,
```

```

    dtype=tf.float32,
    num_parallel_calls=10,
    buffer_size_prefetch=100000,
    buffer_size_shuffle=2048):
def map_func(line):
    columns =tf.string_split(source=[line], delimiter=delimiter, skip_empty=False)
    label =tf.string_to_number(string_tensor=columns.values[0], out_type=tf.int32, name=None)
    feature =tf.string_to_number(string_tensor=columns.values[1: ], out_type=dtype, name=None)
    return feature, label
dataset =tf.data.TextLineDataset(filenames=filenames).\
    map(map_func=map_func, num_parallel_calls=num_parallel_calls).\
    prefetch(buffer_size=buffer_size_prefetch)
if shuffle ==True:
    dataset =dataset.shuffle(buffer_size=buffer_size_shuffle)
dataset =dataset.repeat(count=epochs).batch(batch_size=batch_size, drop_remainder=False)
iterator =dataset.make_one_shot_iterator()
features, labels =iterator.get_next()
return features, labels

def model_fn(features, labels, mode, params):
    """Model function of LeNet5."""
    # -----Define hyper-parameters-----
    num_classes =params["num_classes"]
    image_height =params["image_height"]
    image_width =params["image_width"]
    image_channels =params["image_channels"]
    conv1_size =params["conv1_size"] # The height and width of filters of convolutional layer-1
    conv1_deep =params["conv1_deep"] # The number of filters of convolutional layer-1
    conv2_size =params["conv2_size"]
    conv2_deep =params["conv2_deep"]
    pooling_height =params["pooling_height"]
    pooling_width =params["pooling_width"]
    hidden_units =params["hidden_units"]
    dropout_rates =params["dropout_rates"]
    reg =params["reg"]
    optimizer =params["optimizer"]
    learning_rate =params["learning_rate"]
    dtype =params["dtype"]
    reuse =params["reuse"]

    with tf.variable_scope(name_or_scope="inference", reuse=reuse):
        with tf.variable_scope(name_or_scope="input", reuse=reuse):
            x =tf.reshape(tensor=features, shape=[-1, image_height, image_width, image_channels])

        with tf.variable_scope(name_or_scope="layer1-conv1", reuse=reuse):
            weightsConv1 =tf.get_variable(
                name="weight",
                shape=[conv1_size, conv1_size, image_channels, conv1_deep], # The shape of (filter_height,
                                                                           # filter_width, in_channels, out_channels)
                dtype=dtype,
                initializer=tf.truncated_normal_initializer(mean=0.0, stddev=0.1, dtype=dtype),
                regularizer=l2_regularizer(scale=reg)

```

```

)
biasConv1 =tf.get_variable(
    name="bias",
    shape=[conv1_deep], # The number of filters of convolutional layer-1
    dtype=dtype,
    initializer=tf.constant_initializer(value=0.0, dtype=dtype)
)
conv1 =tf.nn.conv2d(
    input=x,
    filter=weightsConv1,
    strides=[1, 1, 1, 1],
    padding="SAME",
    data_format="NHWC" # it means (None, height, width, channel)
) # A tensor in shape of (None, image_height, image_width, conv1_deep)
relu1 =tf.nn.relu(
    features=tf.nn.bias_add(value=conv1, bias=biasConv1, data_format="NHWC") # tf.nn.bias_add() is
        used for add bias in CNN
) # A tensor in shape of (None, image_height, image_width, conv1_deep)

with tf.variable_scope(name_or_scope="layer2-pool1", reuse=reuse):
    pool1 =tf.nn.max_pool(
        value=relu1,
        ksize=[1, pooling_height, pooling_width, 1],
        strides=[1, pooling_height, pooling_width, 1],
        padding="SAME",
        data_format="NHWC"
) # A tensor in shape of (None, image_height / 2, image_width / 2, conv1_deep)

with tf.variable_scope(name_or_scope="layer3-conv2", reuse=reuse):
    weightsConv2 =tf.get_variable(
        name="weight",
        shape=[conv2_size, conv2_size, conv1_deep, conv2_deep],
        dtype=dtype,
        initializer=tf.truncated_normal_initializer(mean=0.0, stddev=0.1, dtype=dtype),
        regularizer=l2_regularizer(scale=reg)
)
    biasConv2 =tf.get_variable(
        name="bias",
        shape=[conv2_deep],
        dtype=dtype,
        initializer=tf.constant_initializer(value=0.0, dtype=dtype)
)
    conv2 =tf.nn.conv2d(
        input=pool1,
        filter=weightsConv2,
        strides=[1, 1, 1, 1],
        padding="SAME",
        data_format="NHWC"
) # A tensor in shape of (None, image_height / 2, image_width / 2, conv2_deep)
    relu2 =tf.nn.relu(
        features=tf.nn.bias_add(value=conv2, bias=biasConv2, data_format="NHWC")
)

```

```

with tf.variable_scope(name_or_scope="layer4-pool2", reuse=reuse):
    pool2 =tf.nn.max_pool(
        value=relu2,
        ksize=[1, pooling_height, pooling_width, 1],
        strides=[1, pooling_height, pooling_width, 1],
        padding="SAME",
        data_format="NHWC"
    ) # A tensor in shape of (None, image_height / 4, image_width / 4, conv2_deep)
    logits =tf.layers.flatten(inputs=pool2) # Flattens an input tensor while preserving the batch axis
                                                # (axis 0). Note it will return error after
                                                # when using tf.reshape

with tf.variable_scope(name_or_scope="layer5-fc", reuse=reuse):
    for i in xrange(len(hidden_units)):
        logits =tf.layers.dense(
            inputs=logits,
            units=hidden_units[i],
            activation=tf.identity,
            use_bias=True,
            kernel_initializer=tf.glorot_normal_initializer(dtype=dtype),
            kernel_regularizer=l2_regularizer(scale=reg),
            bias_initializer=tf.constant_initializer(value=0.0, dtype=dtype)
        )
        logits =tf.layers.batch_normalization(
            inputs=logits,
            axis=-1,
            momentum=0.99,
            epsilon=1e-3,
            center=True,
            scale=True,
            beta_initializer=tf.zeros_initializer(dtype=dtype),
            gamma_initializer=tf.ones_initializer(dtype=dtype),
            moving_mean_initializer=tf.zeros_initializer(dtype=dtype),
            moving_variance_initializer=tf.ones_initializer(dtype=dtype),
            training=(mode ==tf.estimator.ModeKeys.TRAIN)
        )
        logits =tf.nn.relu(features=logits)
        logits =tf.layers.dropout(
            inputs=logits,
            rate=dropout_rates[i],
            training=(mode ==tf.estimator.ModeKeys.TRAIN)
        ) # A tensor in shape of (None, hidden_units[i])
        logits =tf.layers.dense(
            inputs=logits,
            units=num_classes,
            activation=tf.identity,
            use_bias=True,
            kernel_initializer=tf.glorot_normal_initializer(dtype=dtype),
            kernel_regularizer=l2_regularizer(scale=reg),
            bias_initializer=tf.constant_initializer(value=0.1, dtype=dtype)
        ) # A tensor in shape of (None, num_classes)

```

```

probability =tf.nn.softmax(logits=logits, axis=-1) # A tensor in shape of (None, num_classes)
category =tf.argmax(input=logits, axis=-1) # A tensor in shape of (None, )

predictions ={
    "probability": probability,
    "class": category
}

# -----Provide an estimator spec for `ModeKeys.PREDICTION` mode-----
if mode ==tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)

with tf.variable_scope(name_or_scope="objective", reuse=reuse):
    loss =tf.reduce_mean(
        input_tensor=tf.nn.sparse_softmax_cross_entropy_with_logits(labels=labels, logits=logits), # A
        tensor in shape of (None, )
        axis=0,
        keepdims=False
    )
    regularization =tf.reduce_sum(
        input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
        axis=0,
        keepdims=False
    )
    loss_op =loss +regularization

with tf.variable_scope(name_or_scope="optimization", reuse=reuse):
    if optimizer.lower() == "sgd":
        opt =tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
    elif optimizer.lower() == "momentum":
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=False)
    elif optimizer.lower() == "nesterov":
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=True)
    elif optimizer.lower() == "adagrad":
        opt =tf.train.AdagradOptimizer(learning_rate=learning_rate, initial_accumulator_value=0.1)
    elif optimizer.lower() == "adadelta":
        opt =tf.train.AdadeltaOptimizer(learning_rate=learning_rate, rho=0.95, epsilon=1e-8)
    elif optimizer.lower() == "rmsprop":
        opt =tf.train.RMSPropOptimizer(learning_rate=learning_rate, decay=0.9, epsilon=1e-10)
    elif optimizer.lower() == "adam":
        opt =tf.train.AdamOptimizer(learning_rate=learning_rate, beta1=0.9, beta2=0.999, epsilon=1e-8)
    else:
        raise ValueError("Argument <optimizer> of model_fn is invalid")

    train_op =opt.minimize(loss=loss_op, global_step=tf.train.get_global_step())

# -----Provide an estimator spec for `ModeKeys.TRAIN` mode-----
if (mode ==tf.estimator.ModeKeys.TRAIN):
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

# -----Build evaluation metrics-----
with tf.variable_scope(name_or_scope="evaluation"):

```

```

eval_metric_ops ={
    "accuracy": tf.metrics.accuracy(labels=labels, predictions=category)
}

# -----Provide an estimator spec for `ModeKeys.EVAL` mode-----
if mode ==tf.estimator.ModeKeys.EVAL:
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, predictions=predictions,
                                      eval_metric_ops=eval_metric_ops)

def main(unused_argv):
    task =0
    filenamesTrain =["../../../../data//fashion-mnist_train.csv"]
    filenamesEval =["../../../../data//fashion-mnist_test.csv"]
    filenamesTest =[ "../../../../data//fashion-mnist_test.csv"]
    model_dir ="../../../../log//LeNet5"
    epochs =2000
    batch_size =128
    dtype =tf.float32
    params ={
        "num_classes": 10,
        "image_height": 28,
        "image_width": 28,
        "image_channels": 1,
        "conv1_size": 5,
        "conv1_deep": 32,
        "conv2_size": 5,
        "conv2_deep": 64,
        "pooling_height": 2,
        "pooling_width": 2,
        "hidden_units": [256, 128, 64, 32],
        "dropout_rates": [0.4, 0.3, 0.2, 0.2],
        "reg": 0.003,
        "optimizer": "Adam",
        "learning_rate": 0.001,
        "dtype": dtype,
        "reuse": False
    }
    # Create a estimator
    config =tf.estimator.RunConfig(
        log_step_count_steps=100,
        save_checkpoints_steps=10000,
        keep_checkpoint_max=1
    )
    estimator =tf.estimator.Estimator(model_fn=model_fn, model_dir=model_dir, params=params, config=config)
    # -----Define task-----
    if (task ==0):
        train_spec =tf.estimator.TrainSpec(
            input_fn=lambda :input_fn(
                filenames=filenamesTrain,
                delimiter=",",
                epochs=epochs,

```

```
batch_size=batch_size,
        dtype=dtype,
        shuffle=True
    ),
    max_steps=None
)
eval_spec =tf.estimator.EvalSpec(
    input_fn=lambda :input_fn(
        filenames=filenamesEval,
        delimiter=",",
        epochs=1,
        batch_size=batch_size,
        dtype=dtype,
        shuffle=False
    ),
    steps=None,
    start_delay_secs=120,
    throttle_secs=600
)
tf.estimator.train_and_evaluate(estimator=estimator, train_spec=train_spec, eval_spec=eval_spec)
elif (task ==1):
    estimator.train(
        input_fn=lambda :input_fn(
            filenames=filenamesTrain,
            delimiter=",",
            epochs=epochs,
            batch_size=batch_size,
            dtype=dtype,
            shuffle=True
        ),
        steps=None,
        max_steps=None
    )
elif (task ==2):
    estimator.evaluate(
        input_fn=lambda :input_fn(
            filenames=filenamesEval,
            delimiter=",",
            epochs=1,
            batch_size=batch_size,
            dtype=dtype,
            shuffle=False
        )
    )
elif (task ==3):
    pred_iter =estimator.predict(
        input_fn=lambda :input_fn(
            filenames=filenamesTest,
            delimiter=",",
            epochs=1,
            batch_size=batch_size,
            dtype=dtype,
```

```
        shuffle=True
    )
)
for element in pred_iter:
    print(element["class"])
else:
    raise ValueError("Argument <task> must be in [0, 1, 2, 3]")

if __name__ == '__main__':
    tf.logging.set_verbosity(v=tf.logging.INFO)
    tf.app.run(main=main)
```

16.4 AlexNet

16.4.1 Model formulation

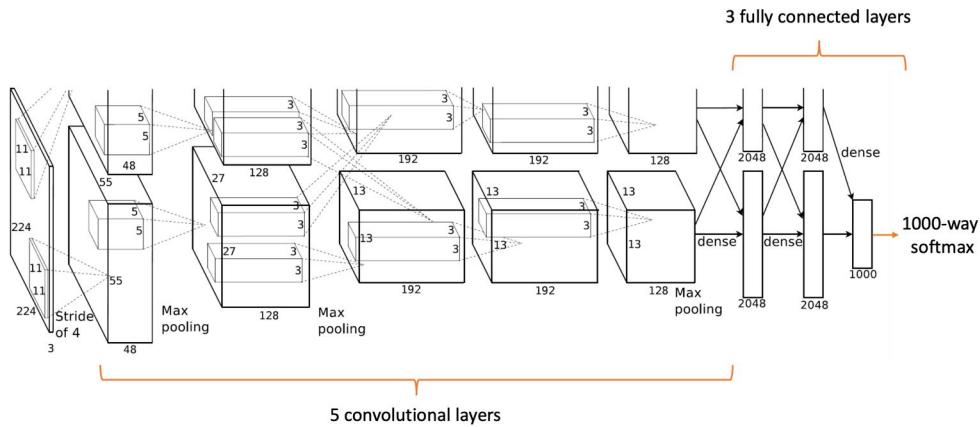


图 54: Architecture of AlexNet[?]

16.4.2 Model implement

TensorFlow 实现代码如下：

```
# Copyright (C) 2019 Tong Jia. All rights reserved.
import tensorflow as tf

def fetch_alexnet_logit_fn(inputs, params, is_training):
    """Definition of the logit inference function (before softmax operation converting into probability
       distribution) of
    AlexNet[Krizhevsky et al., 2012]
    (https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf).
    When in learning phase and use the triplet loss only as loss function, the function output the final layer
    and we
    don't need to add softmax layer after this.

    Parameters
    -----
    :param inputs: (dict) -- containing mini-batch dataset following:
        "images": tensor with shape of (None, 224, 224, num_channels);
        "labels": tensor with shape of (None).
    :param params: (dict) -- containing all hyper-parameters of model.
        "num_classes": (int) -- representing the number of all classes for output the
        distribution.
        "use_dropout": (bool) -- indicating whether to use dropout in all hidden fc layers.
    :param is_training: (bool) -- indicating whether is train phase, or evaluation and inference phases.

    Returns
    -----
    :return: (tensor) -- representing the output tensor with shape of [None, num_classes].
```

```

"""
# Declare all hyper-parameters inside params dict.
num_classes =params["num_classes"]
use_dropout =params["use_dropout"]

# Get the mini-batch images tensor from data dict.
imgs =inputs["images"] # (None, Height = 224, Width = 224, Channel = 3).

with tf.variable_scope(name_or_scope="conv1"):
    # Define trainable variables in 1-th convolution layer.
    in_channels =imgs.get_shape().as_list() [-1]
    out_channels =96
    kernel =tf.get_variable(
        name="kernel",
        shape=[11, 11, in_channels, out_channels],
        dtype=tf.float32,
        initializer=tf.initializers.he_normal()
    )
    bias =tf.get_variable(
        name="bias",
        shape=[out_channels],
        dtype=tf.float32,
        initializer=tf.zeros_initializer(dtype=tf.float32)
    )
    # Define the padding operation.
    paddings =tf.constant(value=[[0, 0], [3, 3], [3, 3], [0, 0]], dtype=tf.int32) # pad shape setting
    xs =tf.pad(tensor=imgs, paddings=paddings, mode="CONSTANT", constant_values=0)
    # Define the convolution operation.
    feature =tf.nn.conv2d(input=xs, filter=kernel, strides=[1, 4, 4, 1], padding="VALID",
                           data_format="NHWC")
    xs =tf.nn.bias_add(value=feature, bias=bias, data_format="NHWC")
    # Define activation function operation.
    ys =tf.nn.relu(features=xs)
    # Define local response normalization (lateral inhibition inside surrounding channels).
    ys =tf.nn.lrn(input=ys, depth_radius=2, bias=2, alpha=1e-4, beta=0.75)

with tf.variable_scope(name_or_scope="pool1"):
    # Define the max pooling operation in 1-th pooling layer.
    xs =tf.nn.max_pool(value=ys, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding="VALID",
                        data_format="NHWC")

with tf.variable_scope(name_or_scope="conv2"):
    # Define trainable variables in 2-th convolution layer. Note only 1-th convolution layer and 2-th
    # convolution
    # layer use local response normalization.
    in_channels =xs.get_shape().as_list() [-1]
    out_channels =256
    kernel =tf.get_variable(
        name="kernel",
        shape=[5, 5, in_channels, out_channels],
        dtype=tf.float32,
        initializer=tf.initializers.he_normal()

```

```

)
bias =tf.get_variable(
    name="bias",
    shape=[out_channels],
    dtype=tf.float32,
    initializer=tf.zeros_initializer(dtype=tf.float32)
)
# Define the padding operation.
paddings =tf.constant(value=[[0, 0], [2, 2], [2, 2], [0, 0]], dtype=tf.int32) # pad shape setting
xs =tf.pad(tensor=xs, paddings=paddings, mode="CONSTANT", constant_values=0)
# Define the convolution operation.
feature =tf.nn.conv2d(input=xs, filter=kernel, strides=[1, 1, 1, 1], padding="VALID",
                      data_format="NHWC")
xs =tf.nn.bias_add(value=feature, bias=bias, data_format="NHWC")
# Define activation function operation.
ys =tf.nn.relu(features=xs)
# Define local response normalization (lateral inhibition inside surrounding channels).
ys =tf.nn.lrn(input=ys, depth_radius=2, bias=2, alpha=1e-4, beta=0.75)

with tf.variable_scope(name_or_scope="pool2"):
    # Define the max pooling operation in 2-th pooling layer.
    xs =tf.nn.max_pool(value=ys, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding="VALID",
                        data_format="NHWC")

with tf.variable_scope(name_or_scope="conv3"):
    # Define trainable variables in 3-th convolution layer.
    in_channels =xs.get_shape().as_list()[-1]
    out_channels =384
    kernel =tf.get_variable(
        name="kernel",
        shape=[3, 3, in_channels, out_channels],
        dtype=tf.float32,
        initializer=tf.initializers.he_normal()
    )
    bias =tf.get_variable(
        name="bias",
        shape=[out_channels],
        dtype=tf.float32,
        initializer=tf.zeros_initializer()
    )
    # Define the padding operation.
    paddings =tf.constant(value=[[0, 0], [1, 1], [1, 1], [0, 0]], dtype=tf.int32) # pad shape setting
    xs =tf.pad(tensor=xs, paddings=paddings, mode="CONSTANT", constant_values=0)
    # Define the convolution operation.
    feature =tf.nn.conv2d(input=xs, filter=kernel, strides=[1, 1, 1, 1], padding="VALID",
                          data_format="NHWC")
    xs =tf.nn.bias_add(value=feature, bias=bias, data_format="NHWC")
    # Define activation function operation.
    ys =tf.nn.relu(features=xs)

with tf.variable_scope(name_or_scope="conv4"):
    # Define trainable variables in 4-th convolution layer. Note between 3-th and 4-th convolution layers,

```

```

    there

# doesn't exist local response normalization and pooling operation.
in_channels =ys.get_shape().as_list()[-1]
out_channels =384
kernel =tf.get_variable(
    name="kernel",
    shape=[3, 3, in_channels, out_channels],
    dtype=tf.float32,
    initializer=tf.initializers.he_normal()
)
bias =tf.get_variable(
    name="bias",
    shape=[out_channels],
    dtype=tf.float32,
    initializer=tf.zeros_initializer()
)
# Define the padding operation.
paddings =tf.constant(value=[[0, 0], [1, 1], [1, 1], [0, 0]], dtype=tf.int32) # pad shape setting
xs =tf.pad(tensor=xs, paddings=paddings, mode="CONSTANT", constant_values=0)
# Define the convolution operation.
feature =tf.nn.conv2d(input=xs, filter=kernel, strides=[1, 1, 1, 1], padding="VALID",
                      data_format="NHWC")
xs =tf.nn.bias_add(value=feature, bias=bias, data_format="NHWC")
# Define activation function operation.
ys =tf.nn.relu(features=xs)

with tf.variable_scope(name_or_scope="conv5"):
    # Define trainable variables in 5-th convolution layer.
    in_channels =ys.get_shape().as_list()[-1]
    out_channels =256
    kernel =tf.get_variable(
        name="kernel",
        shape=[3, 3, in_channels, out_channels],
        dtype=tf.float32,
        initializer=tf.initializers.he_normal()
    )
    bias =tf.get_variable(
        name="bias",
        shape=[out_channels],
        dtype=tf.float32,
        initializer=tf.zeros_initializer()
    )
    # Define the padding operation.
    paddings =tf.constant(value=[[0, 0], [1, 1], [1, 1], [0, 0]], dtype=tf.int32) # pad shape setting
    xs =tf.pad(tensor=xs, paddings=paddings, mode="CONSTANT", constant_values=0)
    # Define the convolution operation.
    feature =tf.nn.conv2d(input=xs, filter=kernel, strides=[1, 1, 1, 1], padding="VALID",
                          data_format="NHWC")
    xs =tf.nn.bias_add(value=feature, bias=bias, data_format="NHWC")
    # Define activation function operation.
    ys =tf.nn.relu(features=xs)

```

```

with tf.variable_scope(name_or_scope="pool5"):
    # Define the max pooling operation in 3-th pooling layer.
    xs = tf.nn.max_pool(value=ys, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding="VALID",
                         data_format="NHWC")
    # Flatten the tensor with shape of (None, Height, Width, Channel).
    xs = tf.layers.flatten(inputs=xs)

with tf.variable_scope(name_or_scope="fc1"):
    if use_dropout:
        # Define dropout operation in 1-th fully connected layer.
        xs = tf.layers.dropout(
            inputs=xs,
            rate=0.2,
            training=is_training
        )
    # Define trainable variables in 1-th fully connected layer.
    in_units = xs.get_shape().as_list()[-1]
    out_units = 4096
    weight = tf.get_variable(
        name="weight",
        shape=[in_units, out_units],
        dtype=tf.float32,
        initializer=tf.initializers.he_normal()
    )
    bias = tf.get_variable(
        name="bias",
        shape=[out_units],
        dtype=tf.float32,
        initializer=tf.zeros_initializer(dtype=tf.float32)
    )
    # Define matrix multiplication operation.
    xs = tf.nn.xw_plus_b(x=xs, weights=weight, biases=bias)
    # Define activation function operation.
    ys = tf.nn.relu(features=xs)

with tf.variable_scope(name_or_scope="fc2"):
    if use_dropout:
        # Define dropout operation in 1-th fully connected layer.
        xs = tf.layers.dropout(
            inputs=xs,
            rate=0.2,
            training=is_training
        )
    # Define trainable variables in 2-th fully connected layer.
    in_units = ys.get_shape().as_list()[-1]
    out_units = 4096
    weight = tf.get_variable(
        name="weight",
        shape=[in_units, out_units],
        dtype=tf.float32,
        initializer=tf.initializers.he_normal()
    )

```

```
bias =tf.get_variable(  
    name="bias",  
    shape=[out_units],  
    dtype=tf.float32,  
    initializer=tf.zeros_initializer(dtype=tf.float32)  
)  
# Define matrix multiplication operation.  
xs =tf.nn.xw_plus_b(x=ys, weights=weight, biases=bias)  
# Define activation function operation.  
ys =tf.nn.relu(features=xs)  
  
with tf.variable_scope(name_or_scope="fc3"):  
    # Define trainable variables in 3-th fully connected layer.  
    in_units =ys.get_shape().as_list() [-1]  
    weight =tf.get_variable(  
        name="weight",  
        shape=[in_units, num_classes],  
        dtype=tf.float32,  
        initializer=tf.variance_scaling_initializer(  
            scale=1.0, mode="fan_avg", distribution="truncated_normal", dtype=tf.float32  
)  
)  
    bias =tf.get_variable(  
        name="bias",  
        shape=[num_classes],  
        dtype=tf.float32,  
        initializer=tf.zeros_initializer(dtype=tf.float32)  
)  
    # Define matrix multiplication operation.  
    logits =tf.nn.xw_plus_b(x=ys, weights=weight, biases=bias) # [None, num_classes]  
  
return logits
```

16.4.3 Reference

- Djordje Slijepcevic: Deep Learning with Tensorflow - AlexNet

16.5 VGG

16.5.1 Model formulation

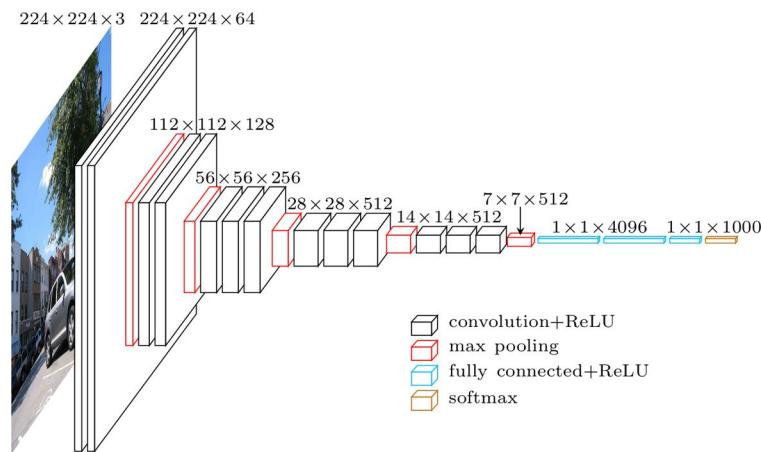


图 55: Architecture of VGG[?]

16.6 GoogLeNet

16.6.1 Model formulation

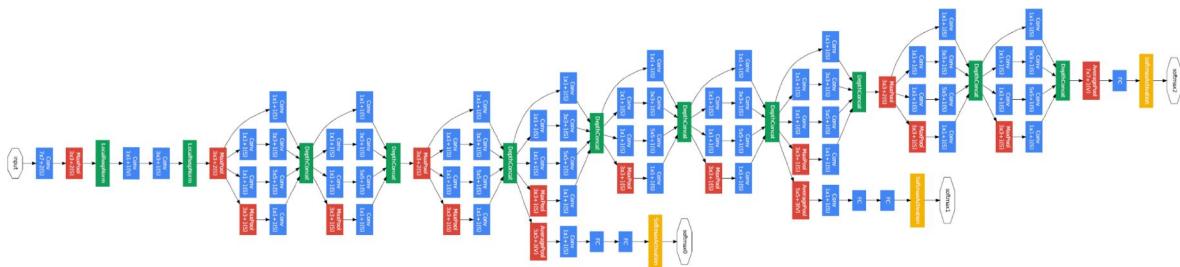


图 56: Architecture of GoogleNet[?]

16.7 ResNet

16.7.1 Model formulation

ResNet[?] (图像领域里程碑式的工作) 的基本单元结构如下：

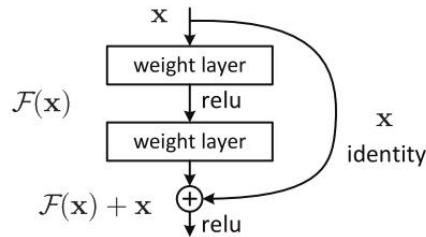


图 57: Architecture of ResNet unit

问题：为何 ResNet 相对于一般的 CNN 可以更有效地防止梯度弥散/梯度爆炸？

$$\begin{aligned}
 \because y &= F(x) + x \\
 \therefore \frac{\partial J}{\partial x} &= \frac{\partial J}{\partial y} \frac{\partial y}{\partial x} \\
 &= \frac{\partial J}{\partial y} \frac{\partial}{\partial x} (F(x) + x) \\
 &= \frac{\partial J}{\partial y} \left(\frac{\partial F(x)}{\partial x} + 1 \right) \\
 &= \underbrace{\frac{\partial J}{\partial y} \frac{\partial F(x)}{\partial x}}_{\text{left-part gradients}} + \underbrace{\frac{\partial J}{\partial y} 1}_{\text{right-part gradients}}
 \end{aligned}$$

注意这里括号内的1为精髓，也就是说为梯度回流开辟了一条 VIP 高速通道。即使单元左侧部分出现梯度弥散： $\frac{\partial F(x)}{\partial x} \simeq 0$ ，单元右侧始终可以保证梯度的有效回流，从而保证了深度模型的有效训练。

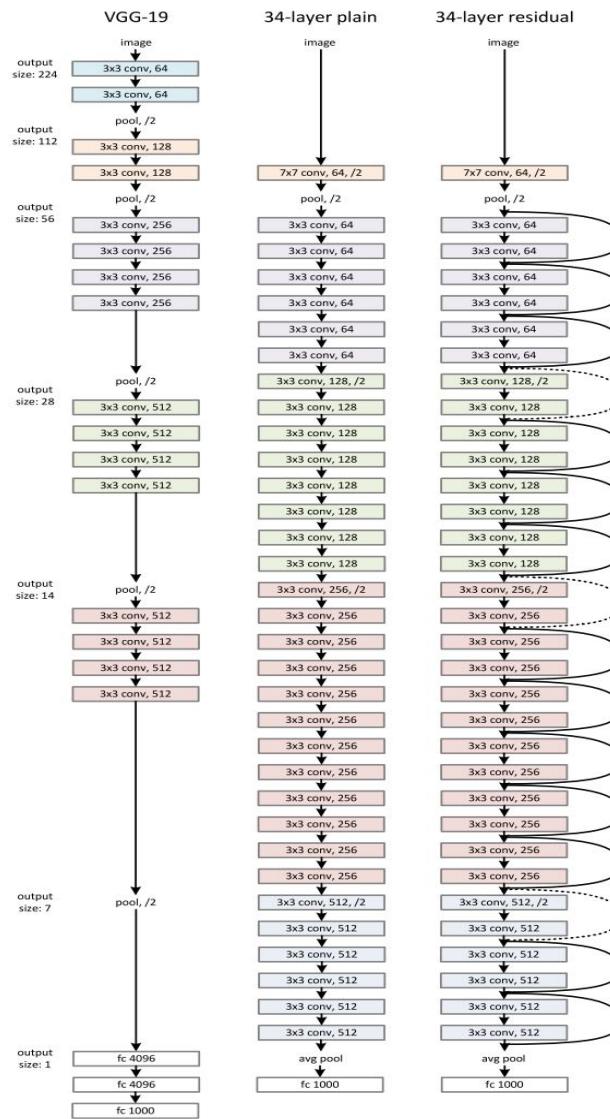


图 58: Architecture of ResNet[?] via architectures of normal CNN

16.8 DenseNet

16.8.1 Model formulation

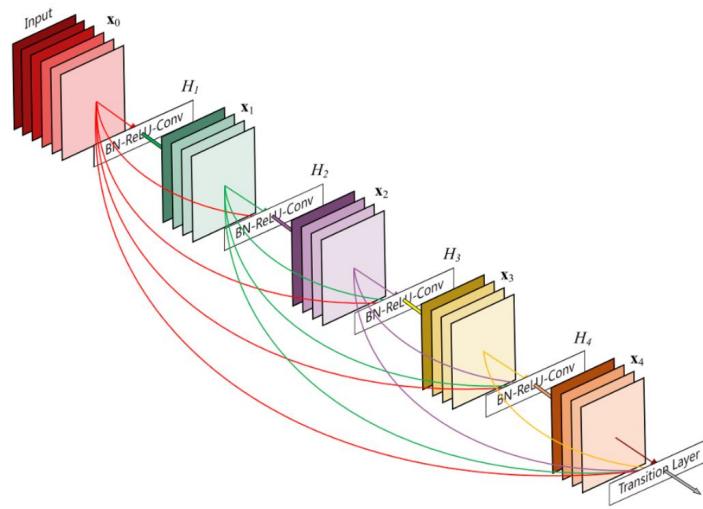


图 59: Architecture of DenseNet[?]

17 Recurrent Neural Networks (RNNs)

17.1 Introduction

RNN 形象化的理解：拼图

1. 眼睛看到现在手里的拼图小块， x_t
2. 回忆目前最新完成的拼图场景， h_{t-1}
3. 结合 1, 2 的信息做出当前时刻拼图小块应该插入到哪里， \hat{y}_t



图 60: Puzzle example for history memory and time series decision

注意第二步，我们在回忆历史的时候，一般不是简单回忆上一个插入的拼图小块，而是去回忆一个**模糊而宏观**的场景，在这个例子中，我们的场景就是当前时刻拼图的整体大概貌，这就是隐层向量 h_{t-1} 。这也是为何不将 h_t 接上一时刻输出 y_{t-1} 的原因。

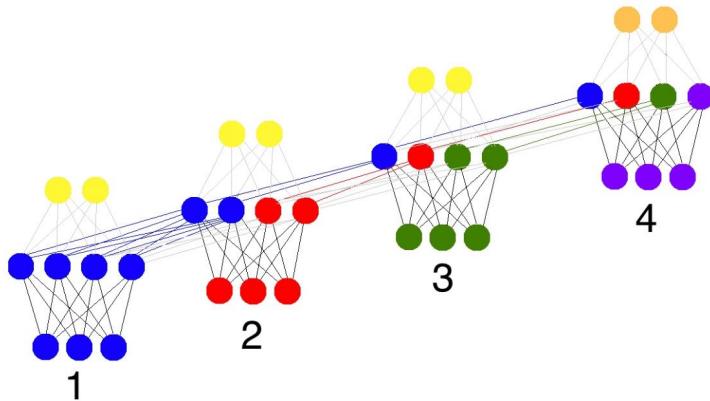


图 61: Recurrent neural network unpack in timestamp

17.2 Vanilla Recurrent Neural Network (RNN)

17.2.1 Derivation of forward-propagation in Vanilla RNN

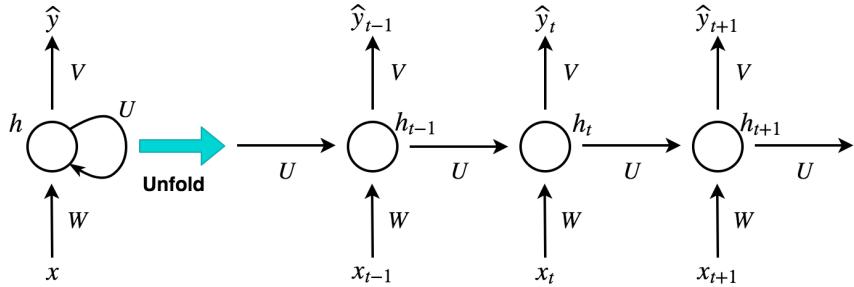


图 62: RNN forward propagation. Image is drawn under [Draw.io](#).

以简单循环网络 (Simple Recurrent Network) 为例进行推导, t 时刻有 :

$$\begin{aligned} h_t &= f(Wx_t + Uh_{t-1} + b) \\ \hat{y}_t &= g(Vh_t) \end{aligned}$$

或者更详细地表示 :

$$\begin{aligned} z_t &= Wx_t + Uh_{t-1} + b \\ h_t &= f(z_t) \\ o_t &= Vh_t \\ \hat{y}_t &= g(o_t) \end{aligned}$$

最后 \hat{y}_t 直接对接了 t 时刻的损失函数 J_t 。参数及变量解释如下 :

- $x_t \in \mathbb{R}^d$: t 时刻的输入向量 (如词向量), 向量长度为 d
- $W \in \mathbb{R}^{l \times d}$
- $U \in \mathbb{R}^{l \times l}$
- $b \in \mathbb{R}^l$

17.2.2 Derivation of back-propagation through time (BPTT) in Vanilla RNN

为了推导的方便, 假设 RNN 在每个时刻 t 都有一个监督信息, 损失函数记为 J_t 。那么整个序列的损失函数记为 :

$$J = \sum_{t=1}^T J_t$$

损失 J 关于参数 U 的梯度为 :

$$\frac{\partial J}{\partial U} = \sum_{t=1}^T \frac{\partial J_t}{\partial U}$$

$$\begin{aligned}
&= \sum_{t=1}^T \frac{\partial J_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial U} \\
&= \sum_{t=1}^T \frac{\partial J_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial U} \\
&= \sum_{t=1}^T \frac{\partial J_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \underbrace{\frac{\partial h_t}{\partial z_t} \left(\sum_{k=1}^t \frac{\partial z_t}{\partial z_k} \frac{\partial z_k}{\partial U} \right)}_{\text{BPTT}} \quad (\because \frac{\partial z_t}{\partial U} = \sum_{k=1}^t \frac{\partial z_t}{\partial z_k} \frac{\partial z_k}{\partial U}) \\
&= \sum_{t=1}^T \sum_{k=1}^t \frac{\partial J_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial U}
\end{aligned}$$

注意上边不要展开求 t 时刻隐层输入到 $t-1$ 时刻隐层输出的梯度 $\frac{\partial z_t}{\partial h_{t-1}}$ ，而是到下面最关键的步骤再展开。

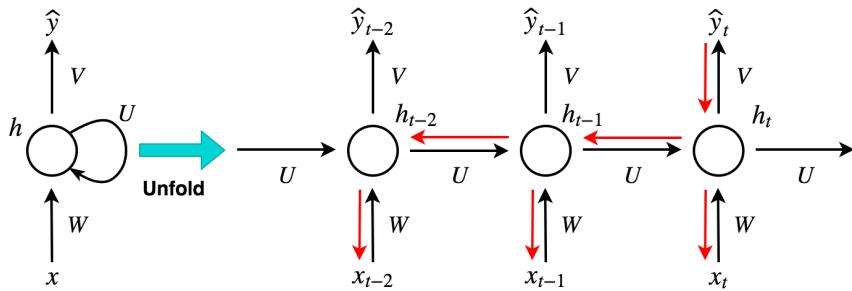


图 63: RNN backpropagation through time

开。即保留 $\frac{\partial z_t}{\partial z_{t-1}}$ 。式子中最关键的梯度：

$$\begin{aligned}
\frac{\partial z_t}{\partial z_k} &= \prod_{i=k+1}^t \frac{\partial z_i}{\partial z_{i-1}} \\
&= \prod_{i=k+1}^t \underbrace{\frac{\partial z_i}{\partial h_{i-1}}}_{\text{易忽略}} \underbrace{\frac{\partial h_{i-1}}{\partial z_{i-1}}}_{\text{易忽略}} \\
&= \prod_{i=k+1}^t U^T \text{diag}(f'(z_{i-1}))
\end{aligned}$$

代入上式中可得：

$$\begin{aligned}
\frac{\partial J}{\partial U} &= \sum_{t=1}^T \sum_{k=1}^t \frac{\partial J_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial z_k} \frac{\partial z_k}{\partial U} \\
&= \sum_{t=1}^T \sum_{k=1}^t \frac{\partial J_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial z_t} \left(\prod_{i=k+1}^t U^T \text{diag}[f'(z_{i-1})] \right) \frac{\partial z_k}{\partial U}
\end{aligned}$$

这样推导出来是最好的展示，因为：

- $\frac{\partial J_t}{\partial \hat{y}_t}$ 只和 t 时刻的损失函数 $J(y_t, \hat{y}_t)$ 有关
- $\frac{\partial \hat{y}_t}{\partial o_t}$ 只和 t 时刻 logit 输出 o_t 与最终输出 \hat{y}_t 之间的激活函数 g 有关

- $\frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t}$ 只和参数 \mathbf{V} 有关
- $\frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t}$ 只和 t 时刻隐层输入 \mathbf{z}_t 与隐层输出 \mathbf{h}_t 之间的激活函数 f 有关

同理可得：

$$\begin{aligned}
 \frac{\partial J}{\partial \mathbf{W}} &= \sum_{t=1}^T \frac{\partial J_t}{\partial \mathbf{W}} \\
 &= \sum_{t=1}^T \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{W}} \\
 &= \sum_{t=1}^T \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \left(\sum_{k=1}^t \frac{\partial \mathbf{z}_t}{\partial \mathbf{z}_k} \frac{\partial \mathbf{z}_k}{\partial \mathbf{W}} \right) \\
 &= \sum_{t=1}^T \sum_{k=1}^t \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{z}_k} \frac{\partial \mathbf{z}_k}{\partial \mathbf{W}} \\
 &= \sum_{t=1}^T \sum_{k=1}^t \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \left(\prod_{i=k+1}^t \mathbf{U}^T \text{diag}[f'(\mathbf{z}_{i-1})] \right) \frac{\partial \mathbf{z}_k}{\partial \mathbf{W}}
 \end{aligned}$$

$$\begin{aligned}
 \frac{\partial J}{\partial \mathbf{b}} &= \sum_{t=1}^T \frac{\partial J_t}{\partial \mathbf{b}} \\
 &= \sum_{t=1}^T \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \hat{\mathbf{o}}_t} \frac{\partial \hat{\mathbf{o}}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{b}} \\
 &= \sum_{t=1}^T \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \left(\sum_{k=1}^t \frac{\partial \mathbf{z}_t}{\partial \mathbf{z}_k} \frac{\partial \mathbf{z}_k}{\partial \mathbf{b}} \right) \\
 &= \sum_{t=1}^T \sum_{k=1}^t \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{z}_k} \frac{\partial \mathbf{z}_k}{\partial \mathbf{b}} \\
 &= \sum_{t=1}^T \sum_{k=1}^t \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \left(\prod_{i=k+1}^t \mathbf{U}^T \text{diag}[f'(\mathbf{z}_{i-1})] \right) \frac{\partial \mathbf{z}_k}{\partial \mathbf{b}}
 \end{aligned}$$

17.2.3 Vanishing gradient problem in RNN

回顾 RNN 中最关键的推导：

$$\begin{aligned}
 \frac{\partial \mathbf{z}_t}{\partial \mathbf{z}_k} &= \prod_{i=k+1}^t \frac{\partial \mathbf{z}_i}{\partial \mathbf{z}_{i-1}} \\
 &= \prod_{i=k+1}^t \frac{\partial \mathbf{z}_i}{\partial \mathbf{h}_{i-1}} \frac{\partial \mathbf{h}_{i-1}}{\partial \mathbf{z}_{i-1}} \\
 &= \prod_{i=k+1}^t \mathbf{U}^T \text{diag}(f'(\mathbf{z}_{i-1}))
 \end{aligned}$$

因此参数 \mathbf{W} , \mathbf{U} 和 \mathbf{b} 的梯度分别为：

$$\frac{\partial J}{\partial \mathbf{W}} = \sum_{t=1}^T \sum_{k=1}^t \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \left(\prod_{i=k+1}^t \mathbf{U}^T \text{diag}[f'(\mathbf{z}_{i-1})] \right) \frac{\partial \mathbf{z}_k}{\partial \mathbf{W}}$$

$$\frac{\partial J}{\partial \mathbf{U}} = \sum_{t=1}^T \sum_{k=1}^t \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \left(\prod_{i=k+1}^t \mathbf{U}^T \text{diag}[f'(\mathbf{z}_{i-1})] \right) \frac{\partial \mathbf{z}_k}{\partial \mathbf{U}}$$

$$\frac{\partial J}{\partial \mathbf{b}} = \sum_{t=1}^T \sum_{k=1}^t \frac{\partial J_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \left(\prod_{i=k+1}^t \mathbf{U}^T \text{diag}[f'(\mathbf{z}_{i-1})] \right) \frac{\partial \mathbf{z}_k}{\partial \mathbf{b}}$$

问题：什么是梯度弥散/梯度爆炸？ 我们定义 $\gamma = \|\mathbf{U}^T \text{diag}(f'(\mathbf{z}_{i-1}))\|$ ，则上述梯度中括号内的部分为 γ^{t-k} ，那么当时刻 k 和时刻 t 相距很远时候：

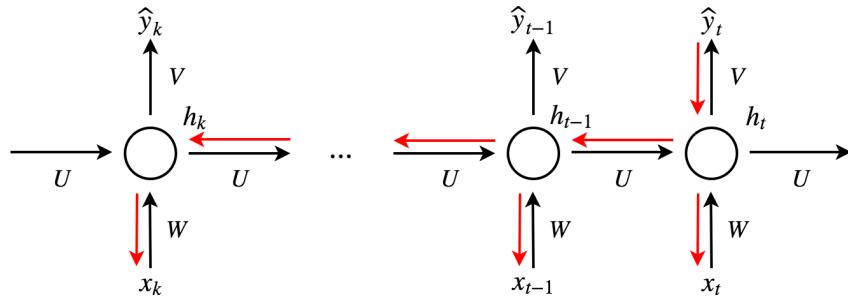


图 64: Vanishing gradient problem in RNN

- $\gamma < 1 \Rightarrow \gamma^{t-k} \simeq 0$: 梯度消失
- $\gamma > 1 \Rightarrow \gamma^{t-k} \rightarrow +\infty$: 梯度爆炸

并且距离越远越明显。

问题：为什么会梯度弥散/梯度爆炸？ 就是 γ 的问题，而 γ 的问题就是激活函数带来的问题。RNN 一般使用 \tanh 函数作为隐层的激活函数，即：

$$\begin{aligned} \because \tanh(x) &= \frac{e^{2x} - 1}{e^{2x} + 1} \\ \therefore \tanh'(x) &= \frac{2e^{2x}(e^{2x} + 1) - (e^{2x} - 1)2e^{2x}}{(e^{2x} + 1)^2} \\ &= \frac{2e^{2x}e^{2x} + 2e^{2x} - 2e^{2x}e^{2x} + 2e^{2x}}{(e^{2x} + 1)^2} \\ &= \frac{4e^{2x}}{(e^{2x} + 1)^2} \\ &= \frac{4e^{2x}}{e^{4x} + 2e^{2x} + 1} \end{aligned}$$

$\tanh(x)$ 和其导数的图像如图所示，可以看到 $\tanh'(x)$ 无论何时都小于 1(除了 $x = 0$)，而权重 $\|\mathbf{U}^T\|$ 也不会太大，我们定义：

- $\|\mathbf{U}^T\| \leq \gamma_u \leq 1$
- $\|\text{diag}[f'(\mathbf{z}_{i-1})]\| \leq \gamma_f \leq 1$

那么可得：

$$\begin{aligned} \because \left\| \frac{\partial z_i}{\partial z_{i-1}} \right\| &= \| U^T \text{diag}[f'(z_{i-1})] \| \leq \| U^T \| \cdot \| \text{diag}[f'(z_{i-1})] \| \leq \gamma_u \gamma_f \leq 1 \\ \therefore \left\| \frac{\partial z_t}{\partial z_k} \right\| &\leq (\gamma_u \gamma_f)^{t-k} \end{aligned}$$

如果间隔 $t - k$ 过大，那么 $\left\| \frac{\partial z_t}{\partial z_k} \right\|$ 会趋于 0，也就是对长距离的依赖无法训练甚至崩溃。

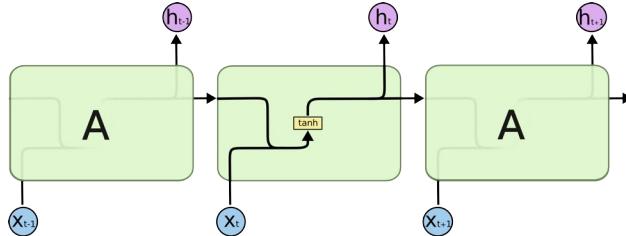


图 65: $\tanh(x)$ is the activation function of RNN hidden unit

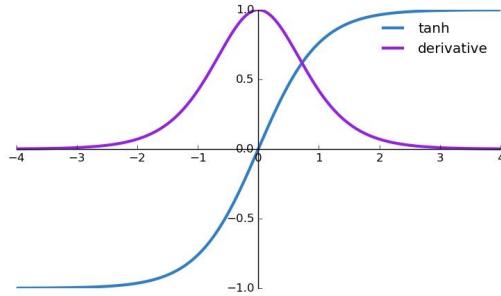


图 66: $\tanh(x)$ and its gradient

问题：梯度弥散会带来什么样的影响？ 例如句子“the clouds are in the sky”这样的预测问题很容易，RNN 就能解决；但是像“ I grew up in France...I speak fluent French.”这样存在句子长期依赖的预测问题，RNN 就会因为梯度弥散而无法解决，这就是所谓的**长期依赖问题**。

17.2.4 References

- Neural Network & Deep Learning
- Understanding LSTM Networks

17.3 Long Short-Term Memory Neural Network (LSTM)

17.3.1 Derivation of forward-propagation in LSTM

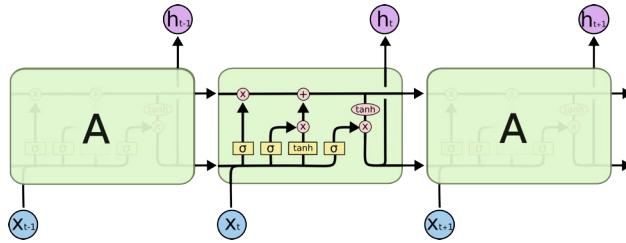
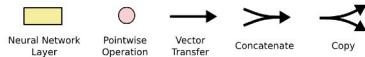
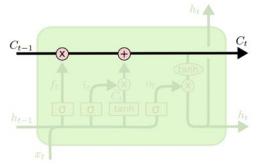


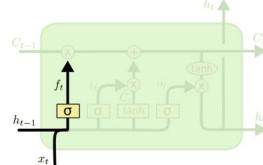
图 67: The repeating module in an LSTM contains four interacting layers.



LSTMs 核心思想 LSTMs[?] 的核心思想是 **cell state**，类似于 **传送带**，决定信息的流量



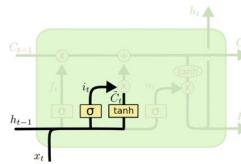
Step1: 决定从 cell state 中扔掉哪些信息



$$f_t = \sigma(\mathbf{W}_f [h_{t-1}, x_t] + \mathbf{b}_f)$$

- 名称：遗忘层（forget gate layer）
- 参数： $\mathbf{W}_f \in \mathbb{R}^{(l+d) \times l}$, $\mathbf{b}_f \in \mathbb{R}^l$ (l 为隐层神经元个数, d 为任意时刻输入向量 x_t 的维度, 下同)
- 输入： $h_{t-1} \in \mathbb{R}^l$, $x_t \in \mathbb{R}^d$
- 输出： $f_t \in \mathbb{R}^l$, $f_{t,j} \in (0, 1)$ (通过 sigmoid 函数激活)
- 作用：稍后作用于 $t - 1$ 时刻的神经元状态 (cell state) C_{t-1} 中的每一个元素 (注：每一个元素 = 每一个隐层神经元)
- 解释：表示保留 C_{t-1} 中每一个元素的程度，即：
 - $f_{t,j} = 1$: 完全保留第 j 个隐层神经元信息
 - $f_{t,j} = 0$: 完全遗忘第 j 个隐层神经元信息

Step2: 决定从 cell state 中保存哪些信息

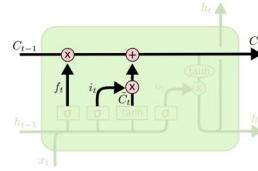


$$i_t = \sigma(\mathbf{W}_i [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

$$\tilde{C}_t = \tanh(\mathbf{W}_C [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C)$$

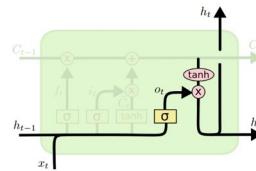
- 名称：输入层 (input gate layer)
- 参数： $\mathbf{W}_i \in \mathbb{R}^{(l+d) \times l}$, $\mathbf{b}_i \in \mathbb{R}^l$, $\mathbf{W}_C \in \mathbb{R}^{(l+d) \times l}$, $\mathbf{b}_C \in \mathbb{R}^l$
- 输入： $\mathbf{h}_{t-1} \in \mathbb{R}^l$, $\mathbf{x}_t \in \mathbb{R}^d$
- 输出： $i_t \in \mathbb{R}^l$, $\tilde{C}_t \in \mathbb{R}^l$
- 作用：稍后会合并二者，对 state 创建一个 update
 - sigmoid 激活函数层：所有隐含神经元记忆新内容的程度
 - tanh 激活函数层：创建一个可能之后会被加载进 state 的候选新内容向量

Step3: 更新旧的 cell state C_{t-1} 为新的 cell state C_t



$$C_t = \underbrace{f_t \odot C_{t-1}}_{\text{对旧信息的记忆}} + \underbrace{i_t \odot \tilde{C}_t}_{\text{对新信息的记忆}}$$

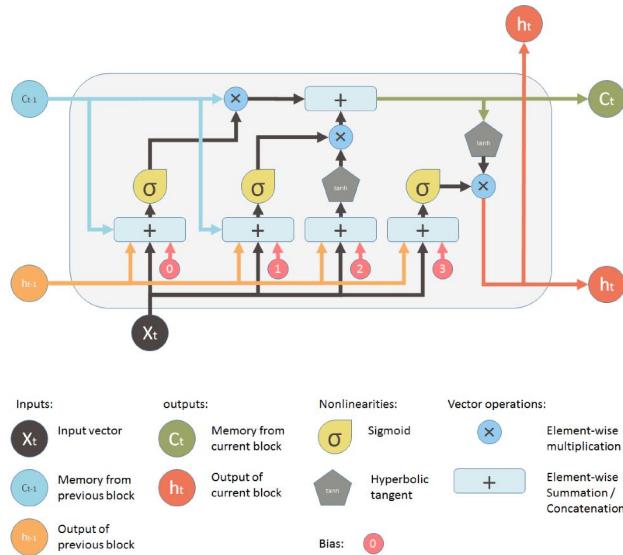
Step4: 依照当前 cell state C_t 决定最终的隐层输出内容



$$o_t = \sigma(\mathbf{W}_o [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$

$$\mathbf{h}_t = o_t \odot \tanh(C_t)$$

综合来看一个 LSTM 的 Unit 如图：



17.3.2 Derivation of back-propagation through time (BPTT) in LSTM

参数解释：

- d : 任意时刻特征向量的维度
- l : 任意时刻隐层向量的维度
- m : Batch size

重新按照深度学习的框架写正向推导：

$$1. \text{ 遗忘门} : \mathbf{f}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{x}_t] \mathbf{W}_f + \mathbf{b}_f)$$

- $\because [\mathbf{h}_{t-1}, \mathbf{x}_t] \in \mathbb{R}^{m \times (l+d)}$
- $\because \mathbf{W}_f \in \mathbb{R}^{(l+d) \times l}$
- $\because \mathbf{b}_f \in \mathbb{R}^{m \times l}$
- $\therefore \mathbf{f}_t \in \mathbb{R}^{m \times l}$

$$2. \text{ 输入门} : \mathbf{i}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{x}_t] \mathbf{W}_i + \mathbf{b}_i), \quad \tilde{\mathbf{C}}_t = \tanh([\mathbf{h}_{t-1}, \mathbf{x}_t] \mathbf{W}_C + \mathbf{b}_C)$$

- $\because [\mathbf{h}_{t-1}, \mathbf{x}_t] \in \mathbb{R}^{m \times (l+d)}$
- $\because \mathbf{W}_i \in \mathbb{R}^{(l+d) \times l}$
- $\because \mathbf{W}_C \in \mathbb{R}^{(l+d) \times l}$
- $\because \mathbf{b}_i \in \mathbb{R}^{m \times l}$
- $\because \mathbf{b}_C \in \mathbb{R}^{m \times l}$
- $\therefore \mathbf{i}_t \in \mathbb{R}^{m \times l}$
- $\therefore \tilde{\mathbf{C}}_t \in \mathbb{R}^{m \times l}$

$$3. \text{ 更新门} : \mathbf{C}_t = \underbrace{\mathbf{f}_t \odot \mathbf{C}_{t-1}}_{\text{对旧信息的记忆}} + \underbrace{\mathbf{i}_t \odot \tilde{\mathbf{C}}_t}_{\text{对新信息的记忆}}$$

- $\because \mathbf{f}_t \in \mathbb{R}^{m \times l}$
- $\because \mathbf{i}_t \in \mathbb{R}^{m \times l}$
- $\because \tilde{\mathbf{C}}_t \in \mathbb{R}^{m \times l}$
- $\because \mathbf{C}_{t-1} \in \mathbb{R}^{m \times l}$
- $\therefore \mathbf{C}_t \in \mathbb{R}^{m \times l}$

$$4. \text{ 输出门} : \mathbf{o}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{x}_t] \mathbf{W}_o + \mathbf{b}_o), \quad \mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$$

- $\because [\mathbf{h}_{t-1}, \mathbf{x}_t] \in \mathbb{R}^{m \times (l+d)}$
- $\because \mathbf{W}_o \in \mathbb{R}^{(l+d) \times l}$
- $\because \mathbf{b}_o \in \mathbb{R}^{m \times l}$
- $\therefore \mathbf{o}_t \in \mathbb{R}^{m \times l}$
- $\because \mathbf{C}_t \in \mathbb{R}^{m \times l}$

- $\therefore \tanh(\mathbf{C}_t) \in \mathbb{R}^{m \times l}$
- $\therefore \mathbf{h}_t \in \mathbb{R}^{m \times l}$

所有待训练参数：

- $\mathbf{W}_f \in \mathbb{R}^{(l+d) \times l}$
- $\mathbf{b}_f \in \mathbb{R}^l$
- $\mathbf{W}_i \in \mathbb{R}^{(l+d) \times l}$
- $\mathbf{W}_C \in \mathbb{R}^{(l+d) \times l}$
- $\mathbf{b}_i \in \mathbb{R}^l$
- $\mathbf{b}_C \in \mathbb{R}^l$
- $\mathbf{W}_o \in \mathbb{R}^{(l+d) \times l}$
- $\mathbf{b}_o \in \mathbb{R}^l$
- $\mathbf{V} \in \mathbb{R}^{l \times 1}$: 隐层到输出层之间的参数

17.3.3 Model implement

```

# -*- coding: utf-8 -*-
"""
Created on 2018/12/17 03:03
author: Tong Jia
email: cecilio.jia@gmail.com
software: PyCharm
Description:
    An implement of single-layer rnn for single-timestep-output regression task.
"""

import tensorflow as tf
from tensorflow.python import rnn
from tensorflow.contrib.layers import l2_regularizer

def input_fn(
    filenames,
    delimiter,
    epochs,
    batch_size,
    dtype,
    shuffle=True,
    num_parallel_calls=10,
    buffer_size_prefetch=100000,
    buffer_size_shuffle=2048):
    def map_func(line):
        columns = tf.string_split(source=[line], delimiter=delimiter, skip_empty=False)
        y = tf.string_to_number(string_tensor=columns.values[0], out_type=dtype)
        x = tf.string_to_number(string_tensor=columns.values[1:], out_type=dtype)
        return x, y
    dataset = tf.data.TextLineDataset(filenames=filenames). \
        map(map_func=map_func, num_parallel_calls=num_parallel_calls). \
        prefetch(buffer_size=buffer_size_prefetch)
    if shuffle ==True:
        dataset =dataset.shuffle(buffer_size=buffer_size_shuffle)
    dataset =dataset. \
        repeat(count=epochs). \
        batch(batch_size=batch_size, drop_remainder=False)
    iterator =dataset.make_one_shot_iterator()
    features, labels =iterator.get_next()
    return features, labels

def model_fn(features, labels, mode, params):
    # -----Define Hyper-parameters-----
    num_steps =params["num_steps"]
    num_hidden_units =params["num_hidden_units"]
    dtype =params["dtype"]
    reg =params["reg"]
    cell =params["cell"] # A string in range of ["RNN", "LSTM", "GRU"]
    optimizer =params["optimizer"]
    learning_rate =params["learning_rate"]

```

```

# -----Define variables-----
with tf.variable_scope(name_or_scope="variables", reuse=tf.AUTO_REUSE):
    W = tf.get_variable(
        name="W",
        shape=[num_hidden_units, 1],
        dtype=dtype,
        initializer=tf.truncated_normal_initializer(mean=0.0, stddev=0.1, dtype=dtype), # adjust
        regularizer=l2_regularizer(scale=reg)
    )
    b = tf.get_variable(
        name="bias",
        shape=[1],
        dtype=dtype,
        initializer=tf.zeros_initializer(dtype=dtype)
    )

with tf.variable_scope(name_or_scope="inference", reuse=tf.AUTO_REUSE):
    x = tf.reshape(tensor=features, shape=[-1, num_steps, 1])
    if cell.lower() == "rnn":
        basicCell = tf.nn.rnn_cell.BasicRNNCell(num_units=num_hidden_units, dtype=dtype)
    elif cell.lower() == "lstm":
        basicCell = tf.nn.rnn_cell.LSTMCell(num_units=num_hidden_units, use_peepholes=False,
                                            forget_bias=1.0,
                                            initializer=tf.orthogonal_initializer(dtype=dtype))
    elif cell.lower() == "gru":
        basicCell = tf.nn.rnn_cell.GRUCell(num_units=num_hidden_units,
                                           kernel_initializer=tf.orthogonal_initializer(dtype=dtype))
    else:
        raise ValueError("Argument <cell> of model_fn is invalid")

    outputs, states = rnn.dynamic_rnn(cell=basicCell, inputs=x, dtype=dtype) # outputs is a tensor in shape
                                                                           # of (None, num_steps, num_inputs)
    outputs = tf.transpose(a=outputs, perm=[1, 0, 2]) # a tensor in shape of (num_steps, None, num_inputs)
    predictions = tf.matmul(a=outputs[-1], b=W) + b # a tensor in shape of (None, 1)
    predictions = tf.reshape(tensor=predictions, shape=[-1])
    pred = {"predictions": predictions}

# -----Provide an estimator spec for `ModeKeys.PREDICTION` mode-----
if mode == tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode, predictions=pred)

with tf.variable_scope(name_or_scope="loss", reuse=tf.AUTO_REUSE):
    loss = tf.losses.mean_squared_error(labels=labels, predictions=predictions)
    regloss = tf.reduce_sum(
        input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
        axis=0,
        keepdims=False
    ) # A scalar representing the regularization loss
    loss += regloss

with tf.variable_scope(name_or_scope="optimizer", reuse=tf.AUTO_REUSE):
    if (optimizer.lower() == "sgd"):
        opt = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)

```

```

    elif (optimizer.lower() == "momentum"):
        opt = tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=False)
    elif (optimizer.lower() == "nesterov"):
        opt = tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=True)
    elif (optimizer.lower() == "adagrad"):
        opt = tf.train.AdagradOptimizer(learning_rate=learning_rate, initial_accumulator_value=0.1)
    elif (optimizer.lower() == "adadelta"):
        opt = tf.train.AdadeltaOptimizer(learning_rate=learning_rate, rho=0.95, epsilon=1e-8)
    elif (optimizer.lower() == "rmsprop"):
        opt = tf.train.RMSPropOptimizer(learning_rate=learning_rate, decay=0.9, epsilon=1e-10)
    elif (optimizer.lower() == "ftrl"):
        opt = tf.train.FtrlOptimizer(learning_rate=learning_rate)
    elif (optimizer.lower() == "adam"):
        opt = tf.train.AdamOptimizer(learning_rate=learning_rate, beta1=0.9, beta2=0.999, epsilon=1e-8)
    else:
        raise ValueError("Argument <optimizer> of model_fn is invalid")
    train_op = opt.minimize(loss=loss, global_step=tf.train.get_global_step())

# -----Provide an estimator spec for `ModeKeys.TRAIN` mode-----
if (mode == tf.estimator.ModeKeys.TRAIN):
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

# -----Build evaluation metrics-----
with tf.variable_scope(name_or_scope="evaluation", reuse=tf.AUTO_REUSE):
    eval_metric_ops = {"rmse": tf.metrics.root_mean_squared_error(labels=labels, predictions=predictions)}

# -----Provide an estimator spec for `ModeKeys.EVAL` mode-----
if mode == tf.estimator.ModeKeys.EVAL:
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, predictions=predictions,
                                      eval_metric_ops=eval_metric_ops)

```

17.4 Gated Recurrent Unit (GRU)

17.4.1 Derivation of forward-propagation in GRU

GRU[?] 相对于标准 LSTMs 的异同：

- 将 LSTM 中的 forget gate 和 input gate 合并为一个 update gate
- 合并 LSTM 中的 cell state 和 hidden state
- 没有了 Bias

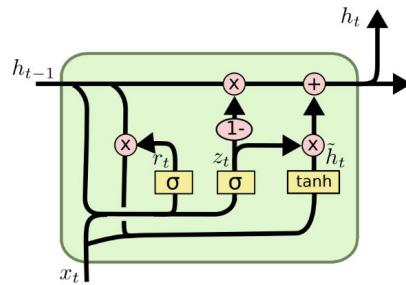


图 68: The unit of GRU[?]

- $z_t = \sigma(W_z [h_{t-1}, x_t])$: 新信息的进入率，老信息的遗忘率
- $r_t = \sigma(W_r [h_{t-1}, x_t])$: 当前时刻对 h_{t-1} 的接受程度
- $\tilde{h}_t = \tanh(W [r_t \odot h_{t-1}, x_t])$: 新生产内容
- $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$: 最终隐含层输出

17.5 Bidirectional LSTM (Bi-LSTM)

17.5.1 Derivation of forward-propagation in Bi-LSTM

双向 LSTM[?] 实质结构为：

- 根据正向的序列数据训练一个 LSTM，在 t 时刻得到正向的隐层输出 $\vec{h}_t \in \mathbb{R}^l$
- 根据反向的序列数据训练一个 LSTM，在 t 时刻得到反向的隐层输出 $\overleftarrow{h}_t \in \mathbb{R}^l$
- 拼接 \vec{h}_t 与 \overleftarrow{h}_t 得到 t 时刻的隐层输出 $h_t = [\vec{h}_t, \overleftarrow{h}_t] \in \mathbb{R}^{2l}$
- 注意**：Bi-LSTM 相对于 LSTM 有更多的信息捕获，因此理论上效果好于 LSTM

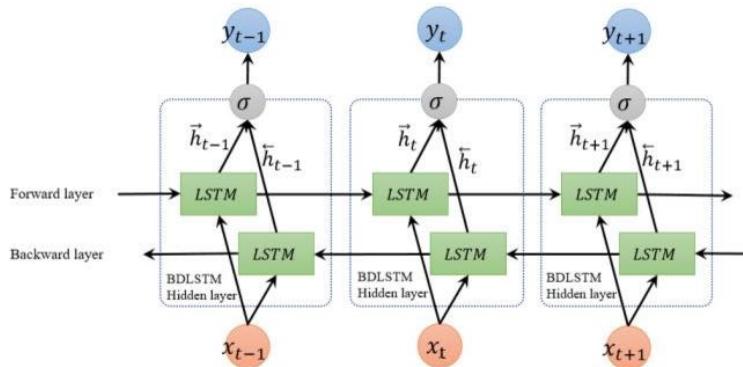


图 69: The structure of bidirectional lstm[?]

注意：此处基本的模型结构 LSTM 也可以替换为 RNN，GRU 等。

17.5.2 Model implement

考虑利用 Bi-LSTM 进行时序销量预测的例子，利用 TensorFlow 实现的代码如下：

```
# -*- coding: utf-8 -*-
"""
Created on 2018/12/19 00:00
author: Tong Jia
email: cecilio.jia@gmail.com
software: PyCharm
Description:
    An implement of single-layer Bidirectional LSTM for single-timestep-output regression task.
"""

import tensorflow as tf
from tensorflow.contrib.layers import l2_regularizer

def input_fn(
    filenames,
    delimiter,
    epochs,
    batch_size,
    dtype,
```

```

shuffle=True,
num_parallel_calls=10,
buffer_size_prefetch=100000,
buffer_size_shuffle=2048):
def map_func(line):
    columns =tf.string_split(source=[line], delimiter=delimiter, skip_empty=False)
    y =tf.string_to_number(string_tensor=columns.values[0], out_type=dtype)
    x =tf.string_to_number(string_tensor=columns.values[1: ], out_type=dtype)
    return x, y

dataset =tf.data.TextLineDataset(filenames=filenames). \
    map(map_func=map_func, num_parallel_calls=num_parallel_calls). \
    prefetch(buffer_size=buffer_size_prefetch)

if shuffle ==True:
    dataset =dataset.shuffle(buffer_size=buffer_size_shuffle)
dataset =dataset. \
    repeat(count=epochs). \
    batch(batch_size=batch_size, drop_remainder=False)
iterator =dataset.make_one_shot_iterator()
features, labels =iterator.get_next()
return features, labels

def model_fn(features, labels, mode, params):
    # -----Define Hyper-parameters-----
    num_steps =params["num_steps"]
    num_hidden_units =params["num_hidden_units"]
    dtype =params["dtype"]
    reg =params["reg"]
    optimizer =params["optimizer"]
    learning_rate =params["learning_rate"]

    # -----Define variables-----
    with tf.variable_scope(name_or_scope="variables", reuse=tf.AUTO_REUSE):
        W =tf.get_variable(
            name="W",
            shape=[2 *num_hidden_units, 1],
            dtype=dtype,
            initializer=tf.truncated_normal_initializer(mean=0.0, stddev=0.1, dtype=dtype), # adjust
            regularizer=l2_regularizer(scale=reg)
        )
        b =tf.get_variable(
            name="bias",
            shape=[1],
            dtype=dtype,
            initializer=tf.zeros_initializer(dtype=dtype)
        )

    with tf.variable_scope(name_or_scope="inference", reuse=tf.AUTO_REUSE):
        x =tf.reshape(tensor=features, shape=[-1, num_steps, 1])
        x =tf.unstack(value=x, num=num_steps, axis=1)
        lstm_cell_fw =tf.nn.rnn_cell.LSTMCell(num_units=num_hidden_units, use_peepholes=False,

```

```

        forget_bias=1.0,
        initializer=tf.orthogonal_initializer(dtype=dtype))
lstm_cell_bw =tf.nn.rnn_cell.LSTMCell(num_units=num_hidden_units, use_peepholes=False,
                                      forget_bias=1.0,
                                      initializer=tf.orthogonal_initializer(dtype=dtype))

outputs, states_fw, states_bw =tf.nn.static_bidirectional_rnn(cell_fw=lstm_cell_fw,
                                                               cell_bw=lstm_cell_bw, inputs=x, dtype=dtype)
# output is a list of tensor with length of num_steps, each element of this list is a tensor in shape
# of (None, 2 * num_hidden_units)
# states_fw & states_bw are tuple of tensor with length of 2, each element of this tuple is a tensor in
# shape of (None, num_hidden_units)

predictions =tf.matmul(a=outputs[-1], b=W) +b
predictions =tf.reshape(tensor=predictions, shape=[-1]) # a tensor in shape of (None, 1)

pred ={"predictions": predictions}

# -----Provide an estimator spec for `ModeKeys.PREDICTION` mode-----
if mode ==tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode, predictions=pred)

with tf.variable_scope(name_or_scope="loss", reuse=tf.AUTO_REUSE):
    loss =tf.losses.mean_squared_error(labels=labels, predictions=predictions)
    regloss =tf.reduce_sum(
        input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
        axis=0,
        keepdims=False
    ) # A scalar representing the regularization loss
    loss +=regloss

with tf.variable_scope(name_or_scope="optimizer", reuse=tf.AUTO_REUSE):
    if (optimizer.lower() == "sgd"):
        opt =tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
    elif (optimizer.lower() == "momentum"):
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=False)
    elif (optimizer.lower() == "nesterov"):
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=True)
    elif (optimizer.lower() == "adagrad"):
        opt =tf.train.AdagradOptimizer(learning_rate=learning_rate, initial_accumulator_value=0.1)
    elif (optimizer.lower() == "adadelta"):
        opt =tf.train.AdadeltaOptimizer(learning_rate=learning_rate, rho=0.95, epsilon=1e-8)
    elif (optimizer.lower() == "rmsprop"):
        opt =tf.train.RMSPropOptimizer(learning_rate=learning_rate, decay=0.9, epsilon=1e-10)
    elif (optimizer.lower() == "ftrl"):
        opt =tf.train.FtrlOptimizer(learning_rate=learning_rate)
    elif (optimizer.lower() == "adam"):
        opt =tf.train.AdamOptimizer(learning_rate=learning_rate, beta1=0.9, beta2=0.999, epsilon=1e-8)
    else:
        raise ValueError("Argument <optimizer> of model_fn is invalid")

train_op =opt.minimize(loss=loss, global_step=tf.train.get_global_step())

```

```
# -----Provide an estimator spec for `ModeKeys.TRAIN` mode-----
if (mode ==tf.estimator.ModeKeys.TRAIN):
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

# -----Build evaluation metrics-----
with tf.variable_scope(name_or_scope="evaluation", reuse=tf.AUTO_REUSE):
    eval_metric_ops ={
        "rmse": tf.metrics.root_mean_squared_error(labels=labels, predictions=predictions)
    }

# -----Provide an estimator spec for `ModeKeys.EVAL` mode-----
```

17.6 Multi Layers Recurrent Neural Networks

17.6.1 Derivation of forward-propagation in Multi-layer RNNs

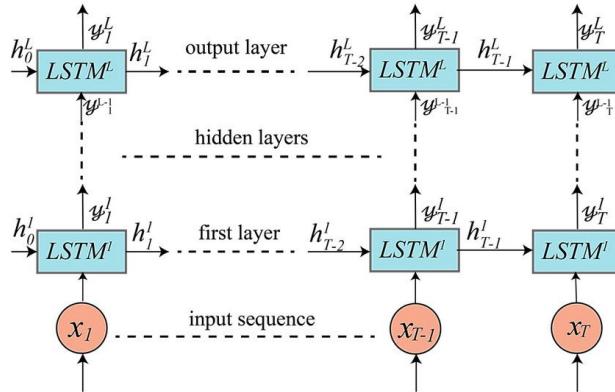


图 70: The structure of multi LSTMs

17.6.2 Model implement

```
# -*- coding: utf-8 -*-
"""
Created on 2018/12/19 01:01
author: Tong Jia
email: cecilio.jia@gmail.com
software: PyCharm
Description:
    An implement of multi-layer LSTMs for single-timestep-output regression task.
"""

import tensorflow as tf
from tensorflow.contrib.layers import l2_regularizer
from tensorflow.python import rnn
from six.moves import xrange

def input_fn(
    filenames,
    delimiter,
    epochs,
    batch_size,
    dtype,
    shuffle=True,
    num_parallel_calls=10,
    buffer_size_prefetch=100000,
    buffer_size_shuffle=2048):
    def map_func(line):
        columns = tf.string_split(source=[line], delimiter=delimiter, skip_empty=False)
        y = tf.string_to_number(string_tensor=columns.values[0], out_type=dtype)
        x = tf.string_to_number(string_tensor=columns.values[1:], out_type=dtype)
        return x, y
    return map_func
```

```

dataset =tf.data.TextLineDataset(filenames=filenames). \
    map(map_func=map_func, num_parallel_calls=num_parallel_calls). \
    prefetch(buffer_size=buffer_size_prefetch)
if shuffle ==True:
    dataset =dataset.shuffle(buffer_size=buffer_size_shuffle)
dataset =dataset. \
    repeat(count=epochs). \
    batch(batch_size=batch_size, drop_remainder=False)
iterator =dataset.make_one_shot_iterator()
features, labels =iterator.get_next()
return features, labels

def model_fn(features, labels, mode, params):
    # -----Define Hyper-parameters-----
    num_steps =params["num_steps"]
    num_hidden_units =params["num_hidden_units"]
    num_hidden_layers =params["num_hidden_layers"]
    dropout_rate =params["dropout_rate"]
    reg =params["reg"]
    optimizer =params["optimizer"]
    learning_rate =params["learning_rate"]
    dtype =params["dtype"]

    # -----Define a function to create a basic rnn cell for multi-layers rnn-----
    def rnn_cell(mode):
        cell =tf.nn.rnn_cell.LSTMCell(
            num_units=num_hidden_units,
            use_peepholes=False,
            initializer=tf.orthogonal_initializer(dtype=dtype),
            forget_bias=1.0,
            state_is_tuple=True
        )
        if mode ==tf.estimator.ModeKeys.TRAIN:
            cell =tf.nn.rnn_cell.DropoutWrapper(
                cell=cell,
                input_keep_prob=1.0,
                output_keep_prob=1 -dropout_rate,
                state_keep_prob=1.0,
                dtype=dtype
            )
        return cell

    # -----Build inference-----
    with tf.variable_scope(name_or_scope="inference", reuse=tf.AUTO_REUSE):
        x =tf.reshape(tensor=features, shape=[-1, num_steps, 1]) # A tensor in shape of (None, num_steps,
                                                               num_inputs)
        batch_size =tf.shape(input=x)[0]
        cells =[rnn_cell(mode=mode) for _ in xrange(num_hidden_layers)] # Note the basic element must be
                                                               created by a function
        multi_cells =tf.nn.rnn_cell.MultiRNNCell(cells=cells, state_is_tuple=True)
        initial_state =multi_cells.zero_state(batch_size=batch_size, dtype=dtype)

```

```

outputs, states =rnn.dynamic_rnn(cell=multi_cells, inputs=x, initial_state=initial_state,
                                 time_major=False)
outputs =tf.transpose(a=outputs, perm=[1, 0, 2]) # output is a tensor in shape of (num_steps, None,
                                         num_hidden_units)

outputs =outputs[-1]

predictions =tf.layers.dense(
    inputs=outputs,
    units=1,
    activation=tf.identity,
    use_bias=True,
    kernel_initializer=tf.truncated_normal_initializer(mean=0.0, stddev=1.0, dtype=dtype),
    bias_initializer=tf.zeros_initializer(dtype=dtype),
    kernel_regularizer=l2_regularizer(scale=reg)
)
predictions =tf.reshape(tensor=predictions, shape=[-1]) # A tensor inshape of (None, )
pred =[{"predictions": predictions}]

# -----Provide an estimator spec for `ModeKeys.PREDICTION` mode-----
if mode ==tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode, predictions=pred)

# -----Build loss-----
with tf.variable_scope(name_or_scope="loss", reuse=tf.AUTO_REUSE):
    loss =tf.losses.mean_squared_error(labels=labels, predictions=predictions)
    regloss =tf.reduce_sum(
        input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
        axis=0,
        keepdims=False
    ) # A scalar representing the regularization loss
    loss +=regloss

# -----Build optimizer-----
with tf.variable_scope(name_or_scope="optimizer", reuse=tf.AUTO_REUSE):
    if (optimizer.lower() == "sgd"):
        opt =tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
    elif (optimizer.lower() == "momentum"):
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=False)
    elif (optimizer.lower() == "nesterov"):
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=True)
    elif (optimizer.lower() == "adagrad"):
        opt =tf.train.AdagradOptimizer(learning_rate=learning_rate, initial_accumulator_value=0.1)
    elif (optimizer.lower() == "adadelta"):
        opt =tf.train.AdadeltaOptimizer(learning_rate=learning_rate, rho=0.95, epsilon=1e-8)
    elif (optimizer.lower() == "rmsprop"):
        opt =tf.train.RMSPropOptimizer(learning_rate=learning_rate, decay=0.9, epsilon=1e-10)
    elif (optimizer.lower() == "ftrl"):
        opt =tf.train.FtrlOptimizer(learning_rate=learning_rate)
    elif (optimizer.lower() == "adam"):
        opt =tf.train.AdamOptimizer(learning_rate=learning_rate, beta1=0.9, beta2=0.999, epsilon=1e-8)
    else:

```

```
    raise ValueError("Argument <optimizer> of model_fn is invalid")

train_op = opt.minimize(loss=loss, global_step=tf.train.get_global_step())

# -----Provide an estimator spec for `ModeKeys.TRAIN` mode-----
if (mode == tf.estimator.ModeKeys.TRAIN):
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

# -----Build evaluation metrics-----
with tf.variable_scope(name_or_scope="evaluation", reuse=tf.AUTO_REUSE):
    eval_metric_ops ={
        "rmse": tf.metrics.root_mean_squared_error(labels=labels, predictions=predictions)
    }

# -----Provide an estimator spec for `ModeKeys.EVAL` mode-----
```

17.7 Bi-LSTM with Multi Layers LSTMs

17.7.1 Forward propagation

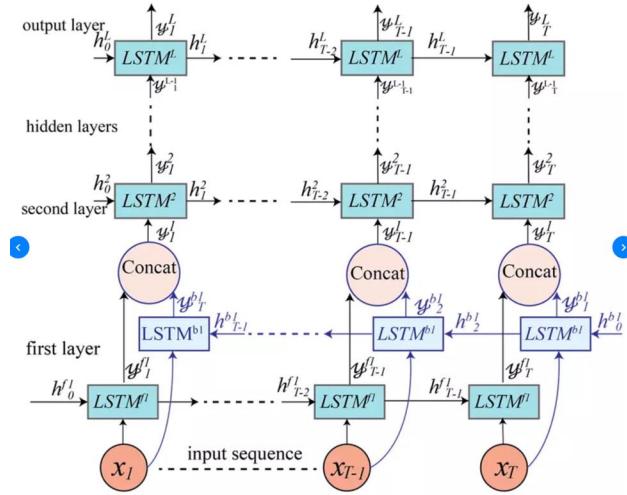


图 71: The stack of BiLSTM with multi LSTMs

18 Sequence to Sequence Learning (Seq2Seq)

18.1 Transformer (未完成)

18.1.1 Introduction

在 NLP 任务中，序列编码是个重要的事情，以 NLP 为例，一般做法是：

1. 句子分词
2. 词 one-hot 编码
3. 转换为词向量

因此每个句子都变成一个矩阵 $\mathbf{X} \in \mathbb{R}^{n \times d}$ 其中：

- n : 句子长度
- d : 词向量维度

现在要做的就是编码 \mathbf{X} ，并从中抽取信息。通用建模方法 (seq2seq 中的 encoder 或 decoder) 包括三类：

- RNN[?]
 - 序列计算限制了模型的并行性
 - 即便是有了 LSTMs 和 GRU，RNN 也是通过注意力机制 (attention mechanism) 来解决长距离依赖的问题
 - 那么一个假设，如果 attention 机制能够帮助我们获取任何的状态 (state)，或许我们就不再需要 RNN
- CNN[?]
- Pure Attention (e.g. Transformer[?])

18.1.2 Model formulation

Transformer[?] 的结构如图所示，包括了 Encoder (左边) 和 Decoder (右边) 两部分。以下将从宏观到微观逐一介绍 Transformer 模型的细节。

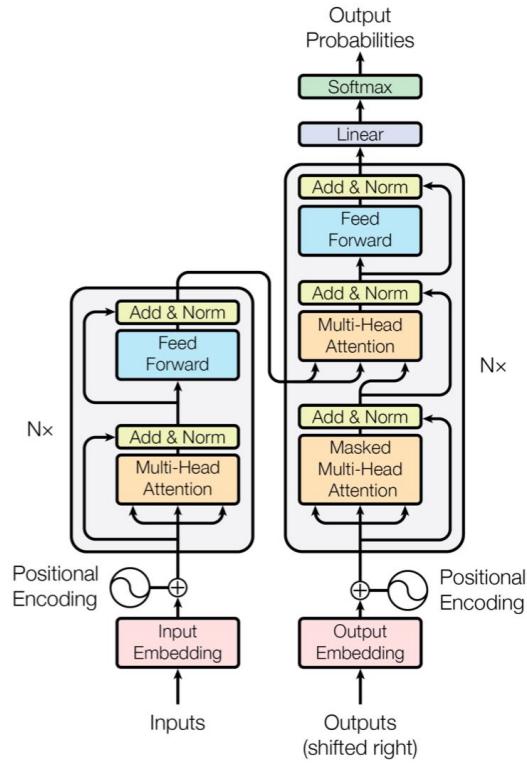


图 72: Architecture of Transformer

微观逐一介绍 Transformer 模型的细节。

A High-Level Look

- 首先可将模型看作一个黑盒子。譬如在机器翻译任务场景中，模型使用一种语言的句子作为输入，并将其翻译为另一种语言作为输出。



图 73: Transformer model for machine translation task as a black box

2. 稍微深入一些，将中间黑盒子拆开，其包括了 Encoders 和 Decoders 两部分结构。

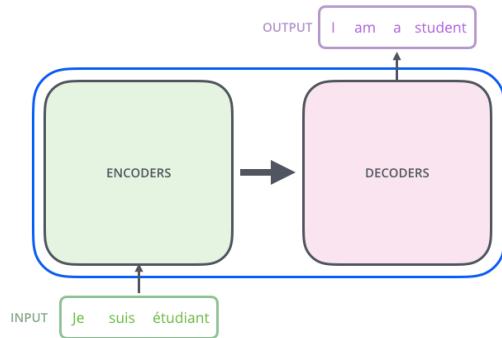


图 74: Unpack the black box as both encoders component and decoders component

3. 再深入一些，Encoders 部分为 N 个完全相同基本 encoder 单元的堆叠，Decoders 部分也是 N 个完全相同基本 decoder 单元的堆叠（论文中设置 $N = 6$ ）。

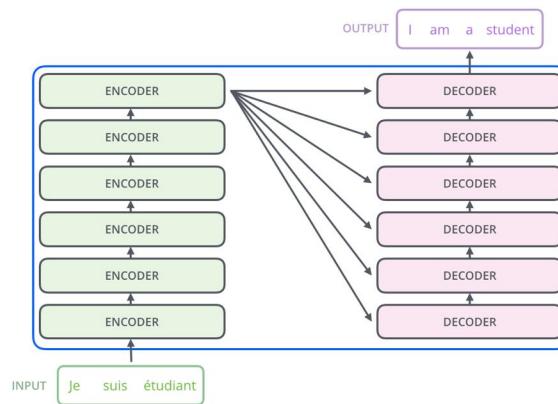


图 75: The encoding component is a stack of encoders, and the decoding component is a stack of decoders with the same number.

4. 进一步深入到基本 encoder 单元结构内部，任意一个基本 encoder 单元由下向上包含两层：Self-Attention 子层和 Feed Forward Neural Network 子层：

- **Self-Attention Layer:** encoder 的输入首先进入 self-attention layer，在编码一个特定单词时，self-attention layer 帮助 encoder **为每一个编码单词“注意”其他单词**。
- **Feed Forward Neural Network Layer:** 获取 self-attention layer 的输出，进行进一步的特征提取。

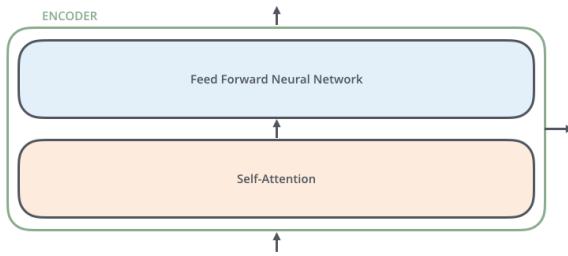


图 76: Basic encoder unit contains two layers: self-attention layer & feed forward nerual netowrk layer

5. 深入到基本 decoder 单元结构内部，稍微不同于基本 encoder 单元，其多了一个 Encoder-Decoder-Attention 层，即由下向上包含了三层：

- **Self-Attention Layer**
- **Encoder-Decoder Attention Layer**
- **Feed Forward Neural Network Layer**

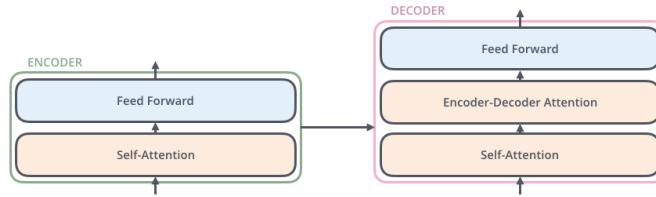


图 77: Basic decoder unit contains three layers: self-attention layer, encoder-decoder attention layer & feed forward nerual netowrk layer

到此为止，已经介绍了 Transformer 模型所有的组成成分，下边将从宏观角度切换到深入细节，跟踪查看 tensors 从模型输入到输出的全过程。

Bringing The Tensors Into The Picture

- 一般 NLP 任务，我们首先需要利用 embedding lookup 将单词转换为向量。



图 78: Each word is embedded into a vector of size 512. We'll represent those vectors with these simple boxes.

- 所有 encoder 的通用抽象输入模式：接收一系列长度为 512 的词向量，其中：

- 最底层的 encoder：接收一系列长度为 512 的 word embeddings

- 其他 encoders：接收一系列上一层 encoder 输出的，长度为 512 的向量
- **注意：**这些“一系列向量”的个数为超参数，我们可以手动设置。一般情况下，该长度为训练数据中最长句子的长度。

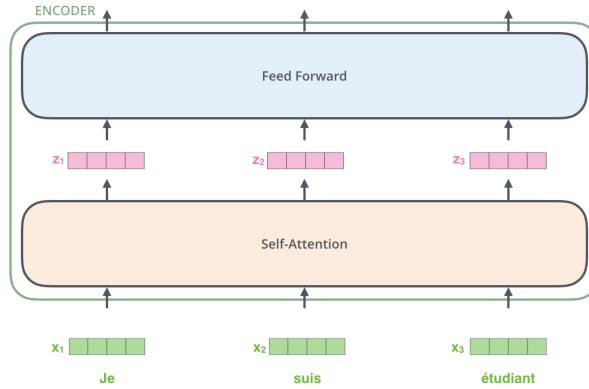


图 79: A list of word embedding flow through two layers of a specific encoder

- Transformer 的一个关键性质：每一个位置的单词（词向量）在 encoder 中其自己独立的路径中流动。因此在 Self-Attention 层中的操作可以并行执行（即可以利用矩阵运算）。

Now Encoding! Tensors 在相邻 encoders 之间的流动

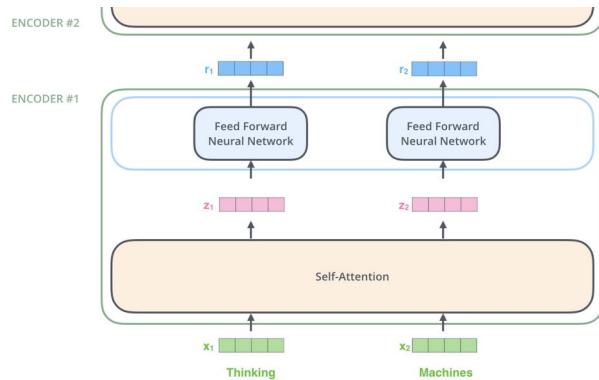


图 80: An encoder receives a list of vectors as input. It processes this list by passing these vectors into a self-attention layer, then into a feed-forward neural network, then sends out the output upwards to the next encoder.

Self-Attention at a High Level

- 考虑以下句子，现在需要对其进行翻译：“The animal didn’t cross the street because it was too tired.”那么句子中的“it”指代 street 还是 animal？这个问题对人类来说很简单，但是对算法而言是有难度的
- 当 Transformer 模型处理单词“it”的时候，self-attention 将关联“it”和“animal”

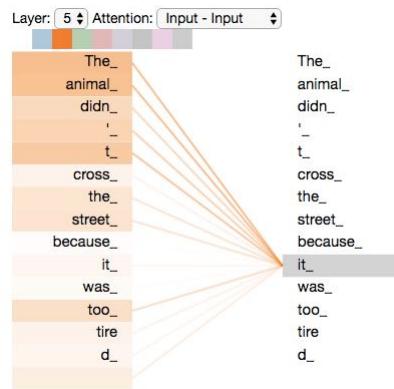


图 81: Self-Attention allows to associate “it”with “animal”

- 当模型处理每一个单词（输入序列（input sequence）中的每个位置（position））时，self-attention 允许它查看输入序列中的其他位置，以寻找可以帮助更好地编码该单词的线索

Self-Attention in Detail 如下先考虑如何利用向量（vectors）计算 self-attention，然后再看实际是如何利用矩阵（matrices）进行计算的。

Self-attention by vector calculation

- 为 encoder 的每个输入向量 \mathbf{X}_j , $j \in \{1, \dots, D\}$ (以 \mathbf{X}_1 为例) 创建三个向量：

- Query vector: $\mathbf{q}_1 = \mathbf{X}_1 \mathbf{W}^Q$
- Key vector: $\mathbf{k}_1 = \mathbf{X}_1 \mathbf{W}^K$
- Value vector: $\mathbf{v}_1 = \mathbf{X}_1 \mathbf{W}^V$

其中 \mathbf{Q} 对应中心词， \mathbf{K} 对应上下文词， \mathbf{V} 对应上下文词。

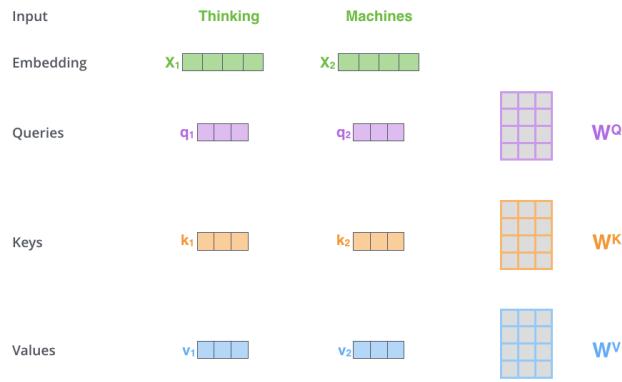


图 82: Create a query vector q_j , a key vector k_j and a value vector v_j for each input vector X_j using matrix multiplication

2. 为每一个位置 (position) 的词 X_j 计算 self-attention。则实际为计算该词对应的 query vector q_j 与句子中所有 key vectors 的内积：

$$\text{Score}(j, j') = q_j \cdot k_{j'}, \quad j' \in \{1, \dots, D\}$$

注意：

- 内积的计算是为了描述中心词特征向量表示 q_j 与所有上下文词特征向量(包含自己)表示 $\{k_1, \dots, k_D\}$ 的相关性。如图可知：Thinking 这个词相对于 Machines 词，其和自身的相关性更大
- $k_{j'}$ 的选取是在 sentence 内的，而不是整个 vocabulary 内

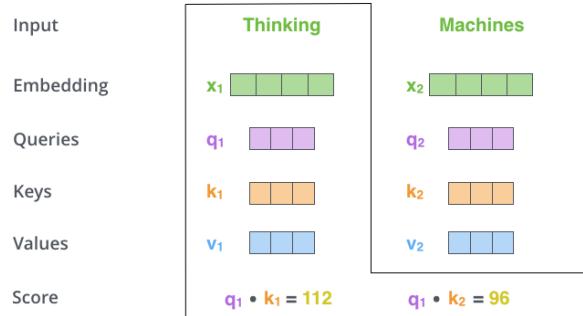


图 83: Calculate self-attention for a specific word in a sentence

3. $\text{Score}(j, j') \leftarrow \frac{\text{Score}(j, j')}{\sqrt{d_k}}$, $j' \in \{1, \dots, D\}$, d_k 为 key vector 的维度.

4. Softmax operation. 作用是使得 $\text{Softmax-score}(j, j') \in (0, 1)$, $j' \in \{1, \dots, D\}$. 该 Softmax 分数 $\text{Softmax-score}(j, j')$ 表达了每个上下文位置 j' 的单词 $X_{j'}$ 在该中心词位置 j 的表达程度

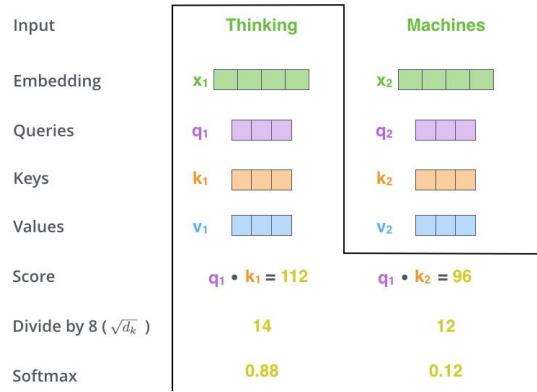


图 84: Softmax score $\text{Score}(j, j')$ determines how much each word $X_{j'}$ will be expressed at this position j

5. $v_j' \leftarrow \text{Softmax-score}(j, j') \times v_{j'}, j' \in \{1, \dots, D\}$. 这里的直觉是:

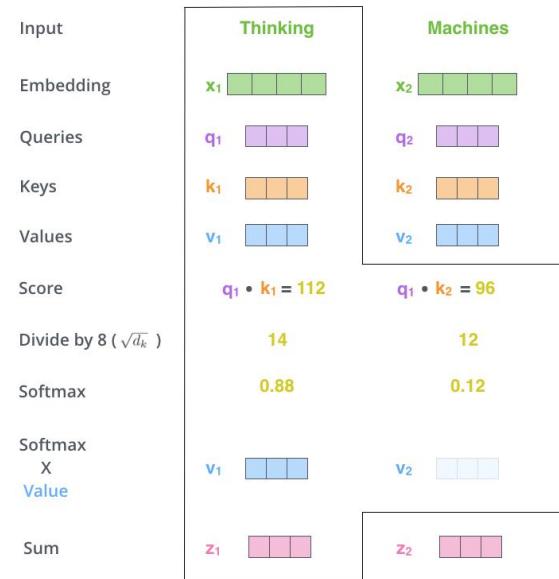


图 85: Multiply each value vector of position j' by the softmax score $\text{Softmax-score}(j, j')$ when processing the central word in position j

- 保持想要关注单词的值的完整性
- 忽略不相关的词 (e.g. 对该上下文词的 value vector 乘以一个很小的 softmax value 譬如 0.001)

6. Sum up the weighted value vectors: $z_j = \sum_{j'=1}^D v_{j'}$. 其中 z_j 表示句子中第 j 个中心词位置，对所有上下文位置 j' 信息容纳表达后的抽象表示。

Self-attention by matrix calculation 上述过程是以向量运算的形式描述 self-attention layer 对一个中心词 position j 的计算过程。但是在实际执行过程中为了高效快速（同时执行多个中心词 position 的 self-attention 操作），这些操作是以矩阵运算的方式执行的，如下将以矩阵运算的方式描述上述过程。

1. Calculate the Query, Key, and Value matrices

- $\because \mathbf{X} \in \mathbb{R}^{D \times E}$ (E is the embedding size (e.g. $E = 512$)), 注意 \mathbf{X} 矩阵中的每一行对应 input sentence 中的一个词
- $\because \mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{E \times H}$ (H is the dimension of query vectors, key vectors and value vectors (e.g. $H = 64$))
- $\therefore \mathbf{Q} = \mathbf{X}\mathbf{W}^Q \in \mathbb{R}^{D \times H}, \mathbf{K} = \mathbf{X}\mathbf{W}^K \in \mathbb{R}^{D \times H}, \mathbf{V} = \mathbf{X}\mathbf{W}^V \in \mathbb{R}^{D \times H}$

The diagram illustrates three separate matrix multiplication operations:

- Top row:** $\mathbf{X} \times \mathbf{W}^Q = \mathbf{Q}$. The input matrix \mathbf{X} (green, 3x3) is multiplied by the weight matrix \mathbf{W}^Q (purple, 3x3) to produce the Query matrix \mathbf{Q} (purple, 3x3).
- Middle row:** $\mathbf{X} \times \mathbf{W}^K = \mathbf{K}$. The input matrix \mathbf{X} (green, 3x3) is multiplied by the weight matrix \mathbf{W}^K (orange, 3x3) to produce the Key matrix \mathbf{K} (orange, 3x3).
- Bottom row:** $\mathbf{X} \times \mathbf{W}^V = \mathbf{V}$. The input matrix \mathbf{X} (green, 3x3) is multiplied by the weight matrix \mathbf{W}^V (blue, 3x3) to produce the Value matrix \mathbf{V} (blue, 3x3).

图 86: Calculate \mathbf{Q} , \mathbf{K} and \mathbf{V} in matrix format

2. Calculate the outputs of the self-attention layer. (由于是矩阵运算，所以可以用一步替代向量运算中的 2-6 步)

The diagram shows the calculation of the output of the self-attention layer:

$$\text{softmax} \left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V} = \mathbf{Z}$$

The input \mathbf{Q} (purple, 3x3) and \mathbf{K}^T (orange, 3x3) are multiplied and then divided by $\sqrt{d_k}$. The result is passed through a softmax function to produce the output \mathbf{Z} (pink, 3x3).

图 87: Calculate the outputs of the self-attention layer

The Beast With Many Heads

- Multi-head attention: 多个 self-attention 组成（论文中总计 8 个）。这样的做法类似与 CNN 中设置多个卷积核，因此可以提供多个表示子空间（multiple representation subspaces），从而提高模型的表达能力。

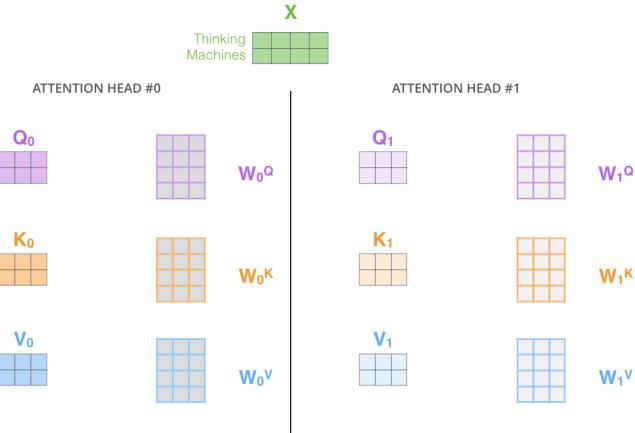


图 88: Multiple \mathbf{W}^Q , \mathbf{W}^K , \mathbf{W}^V correspond to multiple representation subspaces

- 分别计算每个 Attention head :

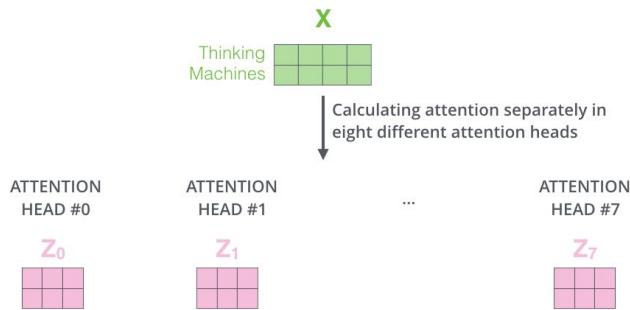
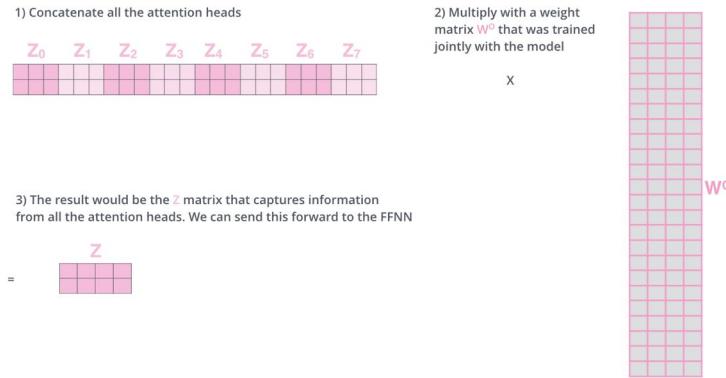


图 89: Calculating each self-attention head separately

- Multi-head attention layer 最后的计算流程如下：

1. Concatenate all attention heads, get a output matrix $[\mathbf{Z}_0, \mathbf{Z}_1, \dots, \mathbf{Z}_{n-1}] \in \mathbb{R}^{D \times (N \times H)}$, N is the number of self-attention
2. Multiply with a weight matrix $\mathbf{W}^O \in \mathbb{R}^{(N \times H) \times E}$
3. get the final output of multi-head attention layer: $\mathbf{Z} \in \mathbb{R}^{D \times E}$

图 90: Concatenate all attention heads, and multiply with a weight matrix $\mathbf{W}^o \in \mathbb{R}^{(n \times E) \times D}$

- Recap of Transformer multi-headed self-attentions:

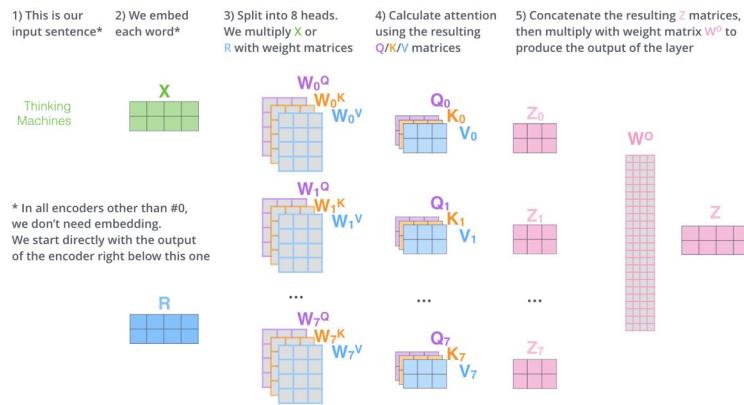


图 91: Transformer multi-headed self-attention recap

Representing The Order of The Sequence Using Positional Encoding 迄今依旧没有考虑的一个问题：输入序列中词的位置信息

18.1.3 References

- The Illustrated Transformer
- The Annotated Transformer
- Blog:The Transformer - Attention is all you need
- Blog of VARUNA JAYASIRI: Transformer in Tensorflow - Attention Is All You Need
- Mac Brennan: Neural Translation Model

18.2 Stabilizing Transformers for Reinforcement Learning (未完成)

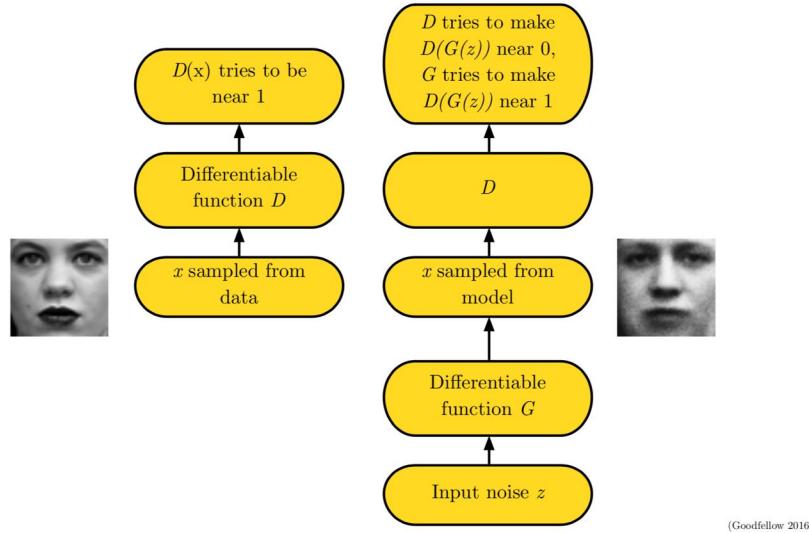
19 Generative Adversarial Networks (GANs)

19.1 Vanilla GAN

19.1.1 Introduction

本文选自 [Generative Adversarial Nets\[?\]](#)，生成对抗网络 (GANs) 的框架如图所示：

Adversarial Nets Framework



(Goodfellow 2016)

图 92: Adversarial Nets Framework[?]

- z : Input noise variables
- $x \in \mathbb{R}^d$: Input data, a high-dimensional vector
- $p_z(z)$: The defined prior distribution on input noise variables
- θ_g : Training parameters of generator G
- θ_d : Training parameters of discriminator D
- $G(z; \theta_g) \in \mathbb{R}^d$: Data space, a high-dimensional vector (e.g. an image, a sequence of words)
- $D(x; \theta_d) \in (0, 1)$: A single scalar output represents the probability that x came from the real data rather than generator G (formulated by sigmoid function)

19.1.2 Model formulation

Build objective function 优化任务：

- Discriminator D : Maximize the probability of assigning the correct label to both training examples and samples from generator G (即：最大化真实数据被 Discriminator 判别为真的概率，同时生成数据被判别为假的概率)

$$\begin{aligned} & \max_D \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[1 - D(G(\mathbf{z}))] \\ \Rightarrow & \max_D \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \end{aligned}$$

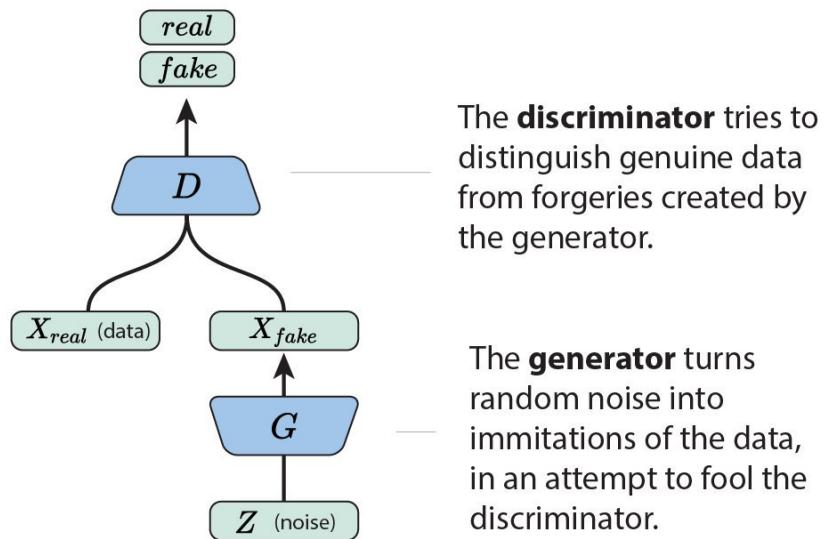
- Generator G : Maximize the probability that the Discriminator D discriminates the generated data $G(\mathbf{z})$ as real data (即：最大化生成数据被判别为真的概率)

$$\begin{aligned} & \max_G \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[D(G(\mathbf{z}))] \\ \Rightarrow & \min_G \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[1 - D(G(\mathbf{z}))] \\ \Rightarrow & \min_G \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \end{aligned}$$

因此同时训练 Discriminator 和 Generator 的目标函数为：

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

Generative Adversarial Networks (GANs) are a way to make a generative model by having two neural networks compete with each other.



Optimization 以 SGD 优化算法为例，模型的优化过程如下所示：

Algorithm 21: Minibatch stochastic gradient descent training of generative adversarial nets[?]

Input: The number of steps k to apply to the discriminator;

Initial training parameters of discriminator θ_d ;

Initial training parameters of generator θ_g ;

Global learning rate of discriminator η_d ;

Global learning rate of generator η_g .

1 **for** *number of training iterations* **do**

2 **for** k *steps* **do**

3 Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$

4 Sample minibatch of m examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution
 $p_{\text{data}}(x)$

5 Update the discriminator by ascending its stochastic gradient:

$$\mathbf{g}_d = -\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D(x^{(i)}) + \log (1 - D(G(z^{(i)}))) \right]$$

$$\theta_d \leftarrow \theta_d - \eta_d \cdot \mathbf{g}_d$$

6 **end**

7 Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$

8 Update the generator by descending its stochastic gradient:

$$\mathbf{g}_g = \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \left[\log (1 - D(G(z^{(i)}))) \right]$$

$$\theta_g \leftarrow \theta_g - \eta_g \cdot \mathbf{g}_g$$

9 **end**

Note:

- 注意在任意 iteration 中是先训练 discriminator，再训练 generator
- 因为只有尽快提高 discriminator（警察）的鉴别能力，才能带动提高 generator（造假者）的造假水平，因此在任意一个 iteration 中，训练 discriminator 多次，而只训练 generator 一次
- 在任意一个 iteration 中，如果只训练 discriminator 一次，而训练 generator 多次，那实际上 discriminator 的鉴别水平不会有太大提高，这时训练 generator 再多次，顶破天也只是达到抗衡当前 discriminator 的水平，所以这时训练 generator 再多次也无意义

Note about GANs

- GANs 系列模型的适用场景：学习给定空间样本分布 \mathcal{Z} 到目标空间样本分布 \mathcal{X} 之间的映射函数，即实现了从 $z \in \mathcal{Z}$ 到 $x \in \mathcal{X}$ 的样本转换
- 在 GAN 的训练中，生成器 Generator 被鼓励产生与真实数据分布 $p_r(x)$ 相似的生成数据分布 $p_g(x)$ ，从而达到产生“以假乱真”生成数据的目的

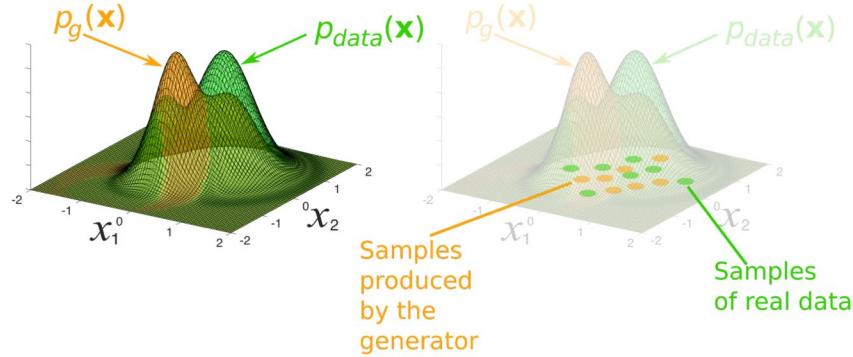


图 93: During GAN training, the generator is encouraged to produce a distribution of samples, $p_g(\mathbf{x})$ to match that of real data, $p_{\text{data}}(\mathbf{x})$. For an appropriately parametrized and trained GAN, these distributions will be nearly identical.[?]

19.2 DCGAN (未完成)

19.2.1 Introduction

本文选自Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks[?]

19.2.2 Model formulation

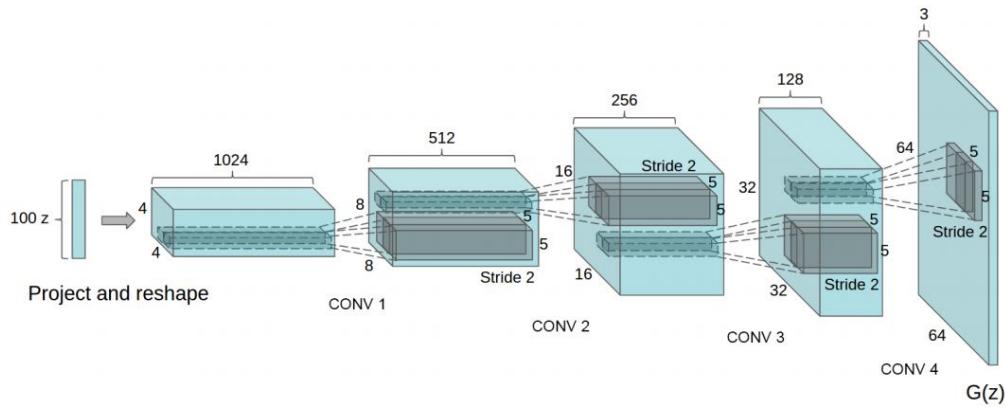


图 94: DCGAN Generator

deconvolution

19.3 Mode Seeking GANs (MSGANs)

19.3.1 Introduction

本文选自Mode Seeking Generative Adversarial Networks for Diverse Image Synthesis[?]，用于解决cGAN中存在的**mode collapse**的问题。同期与之类似的工作还有Diversitysensitive conditional generative adversarial networks[?]

Mode collapse issue for cGANs 原始一系列基于cGAN[?] 的模型存在一个普遍且严重的问题：mode collapse.

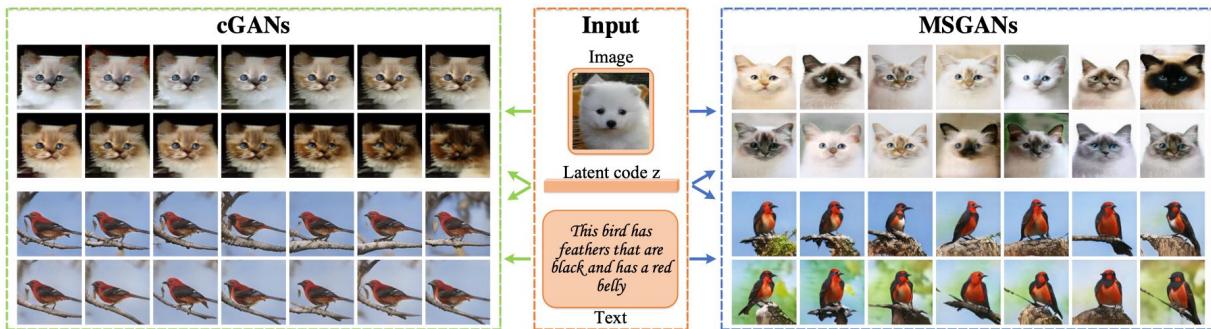


图 95: Mode seeking generative adversarial networks (MSGANs). (Left) Existing conditional generative adversarial networks tend to ignore the input latent code z and generate images of similar modes. (Right) We propose a simple yet effective mode seeking regularization term that can be applied to arbitrary conditional generative adversarial networks in different tasks to alleviate the mode collapse issue and improve the diversity.

原始 cGAN 的目标函数和结构示意图如下：

$$\min_G \max_D V(G, D) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x}|\mathbf{y})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z}|\mathbf{y})))]$$

其中：

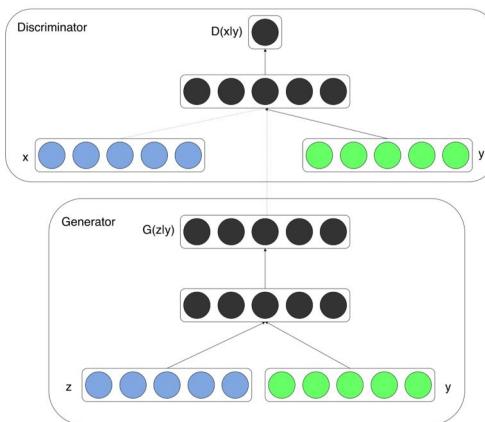


图 96: Conditional adversarial net

- y ，一般称其为 conditional contexts，其可以包含任何形式的辅助信息（auxiliary information），譬如：类别标签和其他形式数据
- 可在 discriminator 和 generator 中将 y 作为附加输入层（additional input layer）以达到执行条件（perform the conditioning）的效果。即：**先验条件就是附加输入层**

尽管 cGANs 已在各种应用中取得了成功，但是现有方法仍遭存在 mode collapse 的问题。这是因为：

1. 条件向量 y 为输出数据（一般是图片）提供了极强的结构先验信息（structural prior information）
2. 条件向量 y 相对于随机噪声向量 z ，拥有更高的特征向量维度

因此导致 generator 更倾向于忽略随机噪声向量 z ，而随机噪声向量 z 正是造就生成图像多样性的原因。因此 generator 忽略随机噪声向量 z 带来结果就是 generator 只产生相似的图片。

Contribution 因此本文的贡献是：提出了一种 mode seeking regularization 的方法用于缓和（alleviate，并非完全解决）cGANs 中存在的 mode collapse 问题，同时该方法是一种通用的方法，而不必像之前的一些做法需要根据特定的任务、特定的数据去修改网络结构（modifications of the network structure）。

19.3.2 Model formulation

Illustration of motivation 本文设计 mode seeking regularization 的动机是：

- 从模式分布（直观理解）的角度看，虽然真实数据中存在很多的 modes，但是当 mode collapse 问题发生时，生成器只会产生很少种类的 modes。
- 从数据分布（微观分析）的角度看，分别计算输入噪声空间 \mathcal{Z} 内两个点 z_a 和 z_b 之间的距离 $d_{\mathcal{Z}}(z_a, z_b)$ ，以及输出样本空间 \mathcal{I} 内两个点 I_a 和 I_b 之间的距离 $d_{\mathcal{I}}(I_a, I_b)$ ，结果通过数值计算发现，当 mode collapse 问题发生时，随着原始空间 \mathcal{Z} 内两个点之间距离的缩短，目标空间 \mathcal{I} 内距离加倍（不成比例）地缩短。换言之，当 $d_{\mathcal{Z}}(z_a, z_b)$ 很大时， $d_{\mathcal{I}}(I_a, I_b)$ 会很小；而当 $d_{\mathcal{Z}}(z_a, z_b)$ 很小时， $d_{\mathcal{I}}(I_a, I_b)$ 会等于 0，因此造成了 mode collapse 的问题。

注意：由上述微观分析可知 L2 距离更容易导致 mode collapse 的问题，因为平方实际上是对差异的放缩。

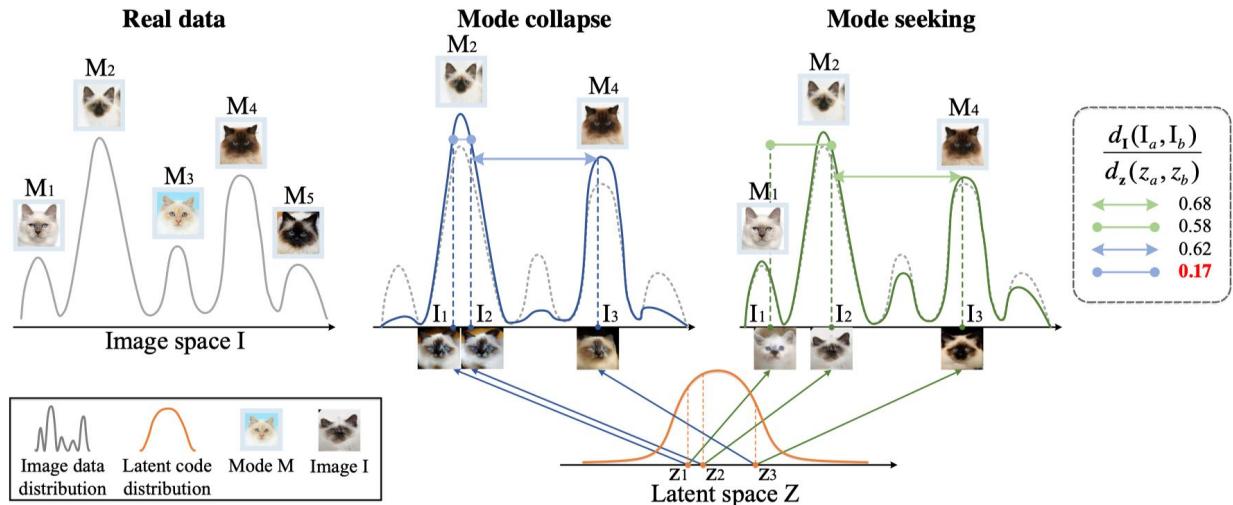


图 97: Real data distribution contains numerous modes. However, when mode collapse occurs, generators only produce samples from a few modes. From the data distribution when mode collapse occurs, we observe that for latent vectors z_1 and z_2 , the distance between their mapped images I_1 and I_2 will become shorter in a disproportionate rate when the distance between two latent vectors is decreasing. We present on the right the ratio of the distance between images with respect to the distance of the corresponding latent vectors, where we can spot an anomalous case (colored in red) where mode collapse occurs. The observation motivates us to leverage the ratio as the training objective explicitly.

因此也就是说两个空间中的距离比 (ratio of the distance) : $\frac{d_{\mathcal{I}}(I_a, I_b)}{d_{\mathcal{Z}}(z_a, z_b)}$ 很小。

Mode seeking regularization 因此针对 Generator 的目标函数中，需要添加如下正则化项，用于最大化 ratio of the distance，尽量缓解 mode collapse 的问题：

$$\begin{aligned} \max_G \mathcal{V}_{\text{ms}}(z_1, z_2) &= \frac{d_{\mathcal{I}}(G(\mathbf{c}, z_1), G(\mathbf{c}, z_2))}{d_{\mathcal{Z}}(z_1, z_2)} \\ \Rightarrow \min_G \mathcal{L}_{\text{ms}}(z_1, z_2) &= -\mathcal{V}_{\text{ms}}(z_1, z_2) \end{aligned}$$

其中：

- $d_*(\cdot)$: distance metric
- c : conditional contexts input

有了此正则化项之后，就增大了 \mathcal{I} 空间内样本的“活动范围”，会有助于生成更多的 mode。譬如图97的例子，对于原始点 z_1 ，不加 mode seeking regularization 时，由于生成分布的活动范围小，因此产生 M_2 模式，而添加 mode seeking regularization 后，得到的对应 I_1 属于 M_1 ，因此相较于之前更大可能产生更多的 mode。

Objective 以 image-to-image translation 任务中的 cGAN 模型 (来自论文Image-to-Image Translation with Conditional Adversarial Networks)[?] 为例，其目标函数为：

$$\min_G \max_D \mathbb{E}_{c,y} [\log D(c, y)] + \mathbb{E}_{c,x} [\log (1 - D(c, G(c, x)))] + \mathbb{E}_{c,y,x} [\|y - G(c, x)\|_1]$$

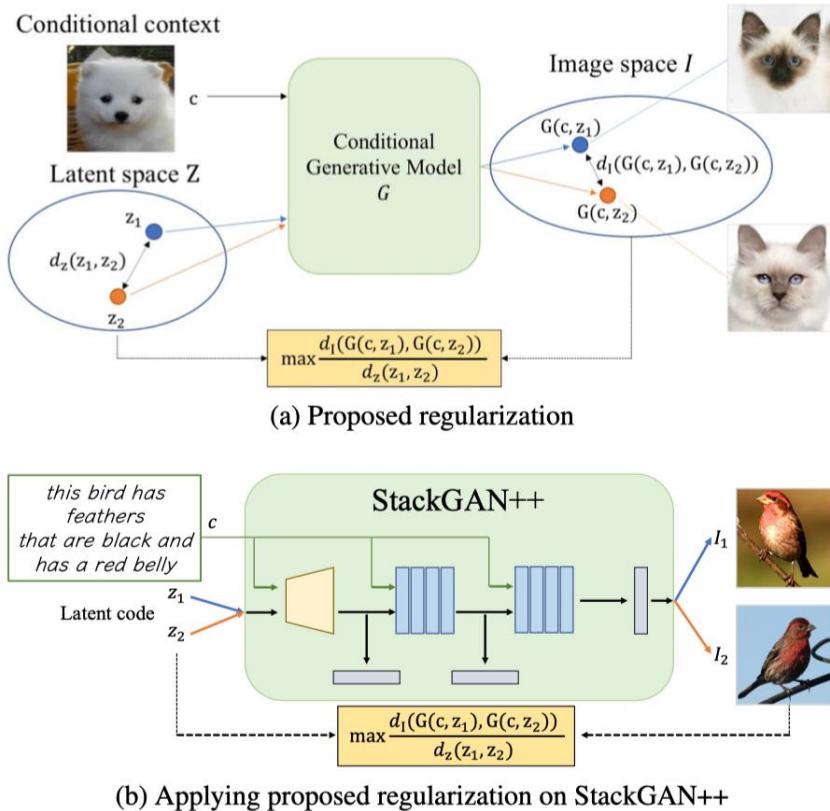


图 98: Proposed regularization. (a) We propose a regularization term that maximizes the ratio of the distance between generated images with respect to the distance between their corresponding input latent codes. (b) The proposed regularization method can be applied to arbitrary cGANs. Take StackGAN++[?], a model for text-to-image synthesis, as an example, we easily apply the regularization term regardless of the complex tree-like structure of the original model.

其中符号表示解释为：

- \mathbf{c} : conditional contexts input(e.g. image of the specific category)
- \mathbf{y} : target domain input
- \mathbf{x} : source domain input

因此在此基础上添加 mode seeking regularization，则模型的目标函数变为（未完成）：

$$\min_G \max_D \mathcal{V}(G, D) = \mathbb{E}_{\mathbf{c}, \mathbf{y}} [\log D(\mathbf{c}, \mathbf{y})] + \mathbb{E}_{\mathbf{c}, \mathbf{x}} [\log (1 - D(\mathbf{c}, G(\mathbf{c}, \mathbf{x})))] + \mathbb{E}_{\mathbf{c}, \mathbf{y}, \mathbf{x}} [\|\mathbf{y} - G(\mathbf{c}, \mathbf{x})\|_1] \\ - \lambda_{\text{ms}} \mathbb{E}_{(\mathbf{x}_1, \mathbf{x}_2)} \mathcal{V}_{\text{ms}}(\mathbf{x}_1, \mathbf{x}_2)$$

注意其中 mode seeking regularization 中在原始空间任意采样样本对满足： $\mathbf{x}_1 \neq \mathbf{x}_2, \forall \mathbf{x}_1, \mathbf{x}_2 \in p_x(\mathbf{x})$

Training procedure

Evaluation metrics and experiments

19.4 Wasserstein GAN (WGAN)

19.4.1 Introduction (未完成)

本文选自 Wasserstein Generative Adversarial Networks[?]，从 GAN 到 WGAN 中使用的概念演化图如下：

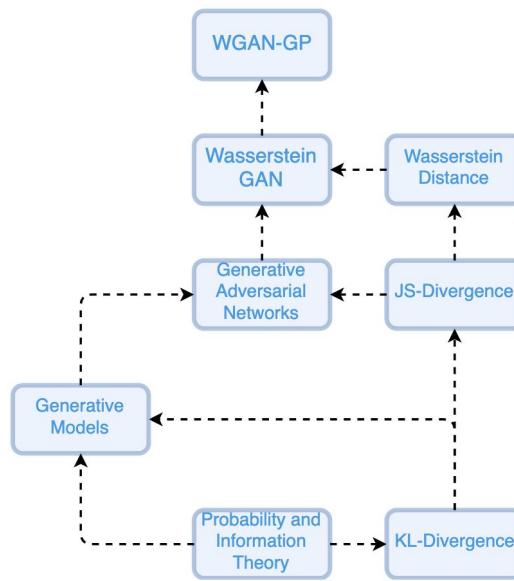


图 99: Concepts used in Wasserstein GAN

The problem of GAN 回顾原始 GAN 模型，模型中存在三种分布：

Symbol	Meaning	Notes
p_z	Data distribution over noise input z	Usually, just uniform.
p_g	The generator's distribution over data x	
p_r	Data distribution over real sample x	

一方面对于 discriminator, 希望 D 在真实样本上的决策可以通过优化以下目标函数而变得准确：

$$\max \mathbb{E}_{x \sim p_r(x)} [\log D(x)]$$

同时在给定假样本 $G(z), z \sim p_z(z)$ 的前提下, D 可以通过优化以下目标函数而达到同样对假样本准确决策的目的：

$$\max \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

另一方面, generator 通过优化以下目标函数被训练, 以达到增强 D 对假样本产生高判别为真概率的目的：

$$\min \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

将这两个方面进行合并, D 和 G 之间实际上是在进行一个 minmax 博弈, 待优化的目标函数如下：

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_r(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

$$= \mathbb{E}_{\mathbf{x} \sim p_r(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_g(\mathbf{x})} [\log (1 - D(\mathbf{x}))]$$

注意，在梯度下降更新过程中 $\mathbb{E}_{\mathbf{x} \sim p_r(\mathbf{x})} [\log D(\mathbf{x})]$ 这一部分对 G 没有影响。

而实际上，GAN 的训练效果并不好，因为其在训练过程中存在以下两个重要的问题：

1. **Vanishing gradient**：判别器 D 判别效果越好，对生成器 G 训练中的梯度消失越严重
2. **Mode collapse**：生成器 G 宁愿生成重复样本，也不愿生成多样性样本

针对以上遇到的这个问题，以下进行理论分析。

Analysis of vanishing gradient problem

What is the optimal value of D ? 因为实际中发现判别器 D 判别效果越好，对生成器 G 训练中的梯度消失越严重，那么就分析在判别器效果最好时的极端情况，因此首先从理论角度找到 D 最优解。

$$\begin{aligned} V(D, G) &= \mathbb{E}_{\mathbf{x} \sim p_r(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_g(\mathbf{x})} [\log (1 - D(\mathbf{x}))] \\ &= \int_{\mathbf{x}} p_r(\mathbf{x}) \log (D(\mathbf{x})) d\mathbf{x} + \int_{\mathbf{x}} p_g(\mathbf{x}) \log (1 - D(\mathbf{x})) d\mathbf{x} \\ &= \int_{\mathbf{x}} (p_r(\mathbf{x}) \log (D(\mathbf{x})) + p_g(\mathbf{x}) \log (1 - D(\mathbf{x}))) d\mathbf{x} \end{aligned}$$

对该分布中任意某一样本情形（概率分布中的一种情况） \mathbf{x} 进行分析，损失函数在该情形上为：

$$V(D, G; \mathbf{x}) = p_r(\mathbf{x}) \log (D(\mathbf{x})) + p_g(\mathbf{x}) \log (1 - D(\mathbf{x}))$$

对其求关于 $D(\mathbf{x})$ 的梯度，即：

$$\frac{\partial V(D, G; \mathbf{x})}{\partial D(\mathbf{x})} = \frac{p_r(\mathbf{x})}{D(\mathbf{x})} + \frac{p_g(\mathbf{x})}{1 - D(\mathbf{x})}$$

令梯度为 0，可得：

$$\begin{aligned} \frac{p_r(\mathbf{x})}{D^*(\mathbf{x})} + \frac{p_g(\mathbf{x})}{1 - D^*(\mathbf{x})} &= 0 \\ \Rightarrow \frac{p_r(\mathbf{x})(1 - D^*(\mathbf{x})) + p_g(\mathbf{x})D^*(\mathbf{x})}{D^*(\mathbf{x})(1 - D^*(\mathbf{x}))} &= 0 \\ \Rightarrow p_r(\mathbf{x})(1 - D^*(\mathbf{x})) + p_g(\mathbf{x})D^*(\mathbf{x}) &= 0 \quad (D^*(\mathbf{x}) \in (0, 1)) \\ \Rightarrow p_r(\mathbf{x}) - p_r(\mathbf{x})D^*(\mathbf{x}) + p_g(\mathbf{x})D^*(\mathbf{x}) &= 0 \\ \Rightarrow p_r(\mathbf{x}) &= (p_r(\mathbf{x}) + p_g(\mathbf{x}))D^*(\mathbf{x}) \\ \Rightarrow D^*(\mathbf{x}) &= \frac{p_r(\mathbf{x})}{p_r(\mathbf{x}) + p_g(\mathbf{x})} \in (0, 1] \quad (p_r(\mathbf{x}) \neq 0 \text{ or } p_g(\mathbf{x}) \neq 0) \end{aligned}$$

即当判别器达到最优时，其对任意样本 \mathbf{x} 的判别输出为 $D^*(\mathbf{x}) = \frac{p_r(\mathbf{x})}{p_r(\mathbf{x}) + p_g(\mathbf{x})}$

What is the loss function when D achieves optimal? 将 $D^*(\mathbf{x}) = \frac{p_r(\mathbf{x})}{p_r(\mathbf{x}) + p_g(\mathbf{x})}$ 带入上述针对任意单个样本的目标函数 $V(D, G; \mathbf{x})$ 中，可得在 D 取得最优时， G 的训练目标函数：

$$V(D^*, G; \mathbf{x}) = p_r(\mathbf{x}) \log (D^*(\mathbf{x})) + p_g(\mathbf{x}) \log (1 - D^*(\mathbf{x}))$$

$$\begin{aligned}
&= p_r(\mathbf{x}) \log \left(\frac{p_r(\mathbf{x})}{p_r(\mathbf{x}) + p_g(\mathbf{x})} \right) + p_g(\mathbf{x}) \log \left(1 - \frac{p_r(\mathbf{x})}{p_r(\mathbf{x}) + p_g(\mathbf{x})} \right) \\
&= p_r(\mathbf{x}) \log \left(\frac{p_r(\mathbf{x})}{p_r(\mathbf{x}) + p_g(\mathbf{x})} \right) + p_g(\mathbf{x}) \log \left(\frac{p_g(\mathbf{x})}{p_r(\mathbf{x}) + p_g(\mathbf{x})} \right) \\
&= p_r(\mathbf{x}) \left(\log(p_r(\mathbf{x})) - \log(p_r(\mathbf{x}) + p_g(\mathbf{x})) \right) + p_g(\mathbf{x}) \left(\log(p_g(\mathbf{x})) - \log(p_r(\mathbf{x}) + p_g(\mathbf{x})) \right) \\
&= p_r(\mathbf{x}) \left(\log(p_r(\mathbf{x})) - \log\left(\frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2} \cdot 2\right) \right) + p_g(\mathbf{x}) \left(\log(p_g(\mathbf{x})) - \log\left(\frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2} \cdot 2\right) \right) \\
&= p_r(\mathbf{x}) \left(\log(p_r(\mathbf{x})) - \log\left(\frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2}\right) - \log 2 \right) + p_g(\mathbf{x}) \left(\log(p_g(\mathbf{x})) - \log\left(\frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2}\right) - \log 2 \right)
\end{aligned}$$

将两个括号中的常数项 $\log 2$ 提出，可得：

$$\begin{aligned}
V(D^*, G; \mathbf{x}) &= p_r(\mathbf{x}) \left(\log(p_r(\mathbf{x})) - \log\left(\frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2}\right) \right) + p_g(\mathbf{x}) \left(\log(p_g(\mathbf{x})) - \log\left(\frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2}\right) \right) \\
&\quad - p_r(\mathbf{x}) \log 2 - p_g(\mathbf{x}) \log 2 \\
&= p_r(\mathbf{x}) \log \left(\frac{p_r(\mathbf{x})}{\frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2}} \right) + p_g(\mathbf{x}) \log \left(\frac{p_g(\mathbf{x})}{\frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2}} \right) - p_r(\mathbf{x}) \log 2 - p_g(\mathbf{x}) \log 2
\end{aligned}$$

因此，当判别器 D 取最优时，GAN 的目标函数为：

$$\begin{aligned}
V(D^*, G) &= \int_{\mathbf{x}} V(D^*, G; \mathbf{x}) d\mathbf{x} \\
&= \int_{\mathbf{x}} p_r(\mathbf{x}) \log \left(\frac{p_r(\mathbf{x})}{\frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2}} \right) d\mathbf{x} + \int_{\mathbf{x}} p_g(\mathbf{x}) \log \left(\frac{p_g(\mathbf{x})}{\frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2}} \right) d\mathbf{x} - \log 2 \int_{\mathbf{x}} p_r(\mathbf{x}) d\mathbf{x} - \log 2 \int_{\mathbf{x}} p_g(\mathbf{x}) d\mathbf{x} \\
&= D_{\text{KL}} \left(p_r(\mathbf{x}) \parallel \frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2} \right) + D_{\text{KL}} \left(p_g(\mathbf{x}) \parallel \frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2} \right) - 2 \log 2 \\
&= 2 \underbrace{\left(\frac{1}{2} D_{\text{KL}} \left(p_r(\mathbf{x}) \parallel \frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2} \right) + \frac{1}{2} D_{\text{KL}} \left(p_g(\mathbf{x}) \parallel \frac{p_r(\mathbf{x}) + p_g(\mathbf{x})}{2} \right) \right)}_{\text{Jensen-Shannon Divergence (JSD)}} - 2 \log 2 \\
&= 2D_{JS} \left(p_r(\mathbf{x}) \parallel p_g(\mathbf{x}) \right) - 2 \log 2
\end{aligned}$$

What is the loss function when D and G both achieve optimal? 因此，一旦生成器 G 也被训练到理想最佳状况下，那么有 $D_{JS} \left(p_r(\mathbf{x}) \parallel p_g(\mathbf{x}) \right) = 0$ ，即 $p_r(\mathbf{x}) = p_g(\mathbf{x})$, $\forall \mathbf{x}$, $D^*(\mathbf{x}) = \frac{1}{2}$ ，则有：

$$\begin{aligned}
V(D^*, G; \mathbf{x}) &= p_r(\mathbf{x}) \log \left(\frac{p_r(\mathbf{x})}{p_r(\mathbf{x})} \right) + p_g(\mathbf{x}) \log \left(\frac{p_g(\mathbf{x})}{p_g(\mathbf{x})} \right) - 2 \log 2 \\
&= p_r(\mathbf{x}) \log 1 + p_g(\mathbf{x}) \log 1 - 2 \log 2 \\
&= -2 \log 2 \quad (\because \log 1 = 0)
\end{aligned}$$

因此当 D 理想最优时，针对 G 的目标函数变为：

$$\begin{aligned}
V(D^*, G) &= \int_{\mathbf{x}} V(D^*, G; \mathbf{x}) d\mathbf{x} \\
&= -2 \log 2 \int_{\mathbf{x}} d\mathbf{x} \\
&= -2 \log 2
\end{aligned}$$

而即使 G 没有完完全全达到理想最佳状况，只是达到近似最佳状况时，也有：

$$p_r(\mathbf{x}) \approx p_g(\mathbf{x})$$

$$\Rightarrow p_r(\mathbf{x}) = p_g(\mathbf{x}) + \epsilon, (\epsilon \text{ is a very small number})$$

则针对某一样本情形 \mathbf{x} 的目标函数可以写为：

$$\begin{aligned} V(D^{\sim*}, G; \mathbf{x}) &= p_r(\mathbf{x}) \log \left(\frac{p_r(\mathbf{x})}{\frac{p_r(\mathbf{x})+p_g(\mathbf{x})-\epsilon}{2}} \right) + p_g(\mathbf{x}) \log \left(\frac{p_g(\mathbf{x})}{\frac{p_g(\mathbf{x})+\epsilon+p_g(\mathbf{x})}{2}} \right) - 2 \log 2 \\ &= p_r(\mathbf{x}) \log \left(\frac{p_r(\mathbf{x})}{p_r(\mathbf{x}) - \frac{\epsilon}{2}} \right) + p_g(\mathbf{x}) \log \left(\frac{p_g(\mathbf{x})}{p_g(\mathbf{x}) + \frac{\epsilon}{2}} \right) - 2 \log 2 \\ &= p_r(\mathbf{x}) \log \left(\frac{p_r(\mathbf{x}) - \frac{\epsilon}{2} + \frac{\epsilon}{2}}{p_r(\mathbf{x}) - \frac{\epsilon}{2}} \right) + p_g(\mathbf{x}) \log \left(\frac{p_g(\mathbf{x}) + \frac{\epsilon}{2} - \frac{\epsilon}{2}}{p_g(\mathbf{x}) + \frac{\epsilon}{2}} \right) - 2 \log 2 \\ &= p_r(\mathbf{x}) \log \left(1 - \frac{\frac{\epsilon}{2}}{p_r(\mathbf{x}) + \frac{\epsilon}{2}} \right) + p_g(\mathbf{x}) \log \left(1 - \frac{\frac{\epsilon}{2}}{p_g(\mathbf{x}) + \frac{\epsilon}{2}} \right) - 2 \log 2 \\ &\approx -2 \log 2 \end{aligned}$$

因此当 D 近似理论最优时，针对 G 的目标函数变为：

$$\begin{aligned} V(D^{\sim*}, G) &= \int_{\mathbf{x}} V(D^{\sim*}, G; \mathbf{x}) d\mathbf{x} \\ &\approx -2 \log 2 \int_{\mathbf{x}} d\mathbf{x} \\ &= -2 \log 2 \end{aligned}$$

因此此时目标函数关于生成器 G 参数 θ_g 的梯度为：

$$\nabla_{\theta_g} V(D^{\sim*}, G) = \nabla_{\theta_g} (-2 \log 2) = 0$$

所以结论是：当判别器 D 达到理论最优，同时生成器 G 达到最优或近似最优时，GAN 的目标函数恒等于或近似等于常数 $-2 \log 2$ ，因此无论采用什么样的优化算法，由于梯度恒为 0，所以生成器 G 的参数永远不会得到更新，即梯度消失 (Vanishing gradient)。

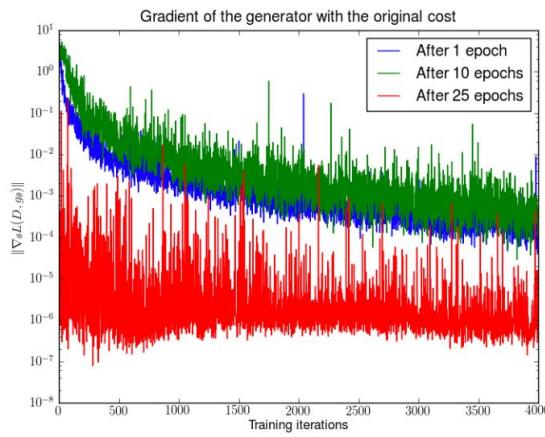


图 100: Gradient value of generator in DCGAN[?]

Analysis of mode collapse problem (未完成)

Improved GAN training (未完成) 以下建议将被用于帮助稳定化 GANs 的训练过程以及提高 GANs 的训练效果，其中：

- 1-5 被实际用于提高 GAN 训练的收敛速度，来自论文：[Improve Techniques for Training GANs\[?\]](#)
- 6-7 被用于解决不相交分布 (disjoint distributions) 的问题，来自论文：[Towards principled methods for training generative adversarial networks\[?\]](#)

这些方法如下：

- Feature Matching
- Minibatch Discrimination
- Historical Averaging
- One-sided Label Smoothing
- Virtual Batch Normalization(VBN)
- Adding Noises
- Use Better Metric of Distribution Similarity

19.4.2 Wasserstein distance and the Kantorovich-Rubinstein duality (未完成)

19.4.3 Model formulation

Objective WGAN 最终的目标函数为：

$$\min_{\theta} \max_{\omega} \mathbb{E}_{x \sim \mathbb{P}_r} [f_{\omega}(x)] - \mathbb{E}_{z \sim p(z)} [f_{\omega}(g_{\theta}(z))]$$

其中判别器的判别函数 $f_{\omega}(\cdot) \in \mathbb{R}$, 而非原始 GAN 中的 $f_{\omega}(\cdot) \in (0, 1)$.

Training procedure 训练算法的伪代码如下：

Algorithm 22: Wasserstein GAN[?]

Input: α : the learning rate (defaults to 0.00005);
 c : the clipping parameter (defaults to 0.01);
 m : the batch size (defaults to 64);
 n_{critic} : the number of iterations of the critic per generator iteration (defaults to 5).

Input: ω : initial critic parameters;
 θ : initial generator's parameters.

Output: The final parameters of generator θ

```

1 while  $\theta$  has not converged do
2   for  $t = 0, \dots, n_{\text{critic}}$  do
3     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from real data.
4     Sample  $\{z^{(i)}\}_{i=1}^m \sim p_z(z)$  a batch of prior samples.
5     Compute update direction for critic:  $\Delta_{\omega} \leftarrow \nabla_{\omega} \left[ \frac{1}{m} \sum_{i=1}^m f_{\omega}(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_{\omega}(g_{\theta}(z^{(i)})) \right]$ 
6     Updata critic parameters:  $\omega \leftarrow \omega + \text{RMSProp}(\omega, \Delta_{\omega}, \alpha)$ 
7     Clipping for critic parameters:  $\omega \leftarrow \text{clip}(\omega, -c, c)$ 
8   end
9   Sample  $\{z^{(i)}\}_{i=1}^m \sim p_z(z)$  a batch of prior samples.
10  Compute update direction for generator:  $\Delta_{\theta} \leftarrow \nabla_{\theta} \frac{1}{m} \sum_{i=1}^m f_{\omega}(g_{\theta}(z^{(i)}))$ 
11  Updata generator's parameters:  $\theta \leftarrow \theta + \text{RMSProp}(\theta, \Delta_{\theta}, \alpha)$ 
12 end

```

注意：

- critic(discriminator) 的优化为 max, 所以更新方向为梯度 **正方向**
- generator 优化为 min, 所以更新方向为梯度 **负方向**, 或者直观来看, 其目标函数可以转换为:

$$\max_{\theta} \mathbb{E}_{z \sim p(z)} [f_{\omega}(g_{\theta}(z))]$$

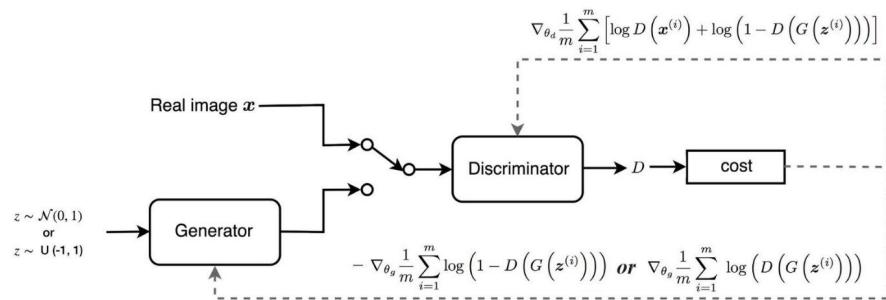
因此更新方向就是梯度方向, 即 $\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m f_{\omega}(g_{\theta}(z^{(i)}))$

Summary WGAN 和 GAN 的区别：

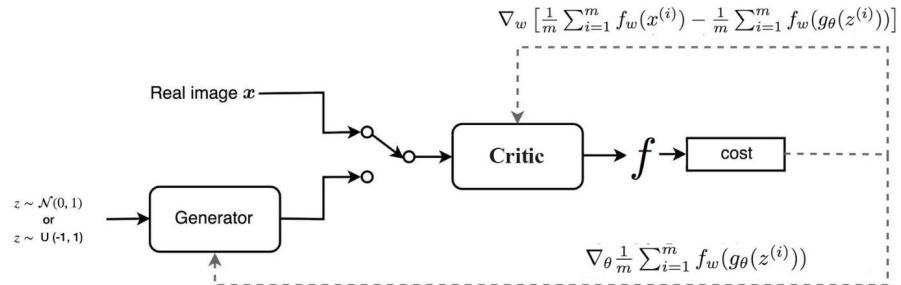
- WGAN 的 Critic(Discriminator) 映射关系为 $f_{\omega}(\cdot) : \mathbb{R}^{\mathcal{X}} \rightarrow \mathbb{R}$, 而 GAN (Vanilla GAN) 中为 $f_{\omega}(\cdot) : \mathbb{R}^{\mathcal{X}} \rightarrow (0, 1)$, 也就是说 WGAN 最后一层没有 sigmoid 激活函数

- WGAN 的 Critic(Discriminator) 输出函数没有 log 项
- WGAN 的 Critic 在每次更新后都要把参数截断在某个范围，即 weight clipping，这是为了保证上面讲到的 Lipschitz 限制
- Critic 训练得越好，对 Generator 的提升更有利，因此可以放心地多训练 Critic，而 Vanilla GAN 的 Discriminator 训练的越好，对 Generator 的提升更不利
- WGAN 和 GAN 的模型对比图如下：

GAN:



WGAN



19.4.4 Reference

- From GAN to WGAN by Lil'Log
- Wasserstein GAN by James Allingham
- Read-through: Wasserstein GAN
- An intuitive guide to optimal transport, part I: formulating the problem
- GAN—Wasserstein GAN & WGAN-GP by Jonathan Hui

19.5 Wasserstein GAN with Gradient Penalty (WGAN-GP)

19.5.1 Introduction

本文选自 Improved Training of Wasserstein GANs[?]

19.5.2 Model formulation

Objective 模型的目标函数如下：

$$\min_{\theta} \max_{\omega} V(\theta, \omega) = \mathbb{E}_{x \sim \mathbb{P}_r} [D(x)] - \mathbb{E}_{\tilde{x} \sim \mathbb{P}_g} [D(\tilde{x})] - \lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} [(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]$$

其中：

$$\hat{x} = \epsilon x + (1 - \epsilon) \tilde{x}$$

Training procedure 训练算法的伪代码如下：

Algorithm 23: WGAN with gradient penalty[?]

Input: λ : the gradient penalty coefficient;

n_{critic} : the number of iterations of the critic per generator iteration (defaults to 5);

m : the batch size (defaults to 64);

α, β_1, β_2 : Adam hyperparameters.

Input: ω : initial critic parameters;

θ : initial generator's parameters.

Output: The final parameters of generator θ

```

1 while  $\theta$  has not converged do
2   for  $t = 0, \dots, n_{critic}$  do
3     for  $i = 1, \dots, m$  do
4       Sample real data  $x \sim \mathbb{P}_r$ , latent variable  $z \sim p(z)$ , a random number  $\epsilon \sim U[0, 1]$ .
5       Compute the output of generator:  $\tilde{x} \leftarrow G_{\theta}(z)$ 
6       Construct a noise-convex sample:  $\hat{x} \leftarrow \epsilon x + (1 - \epsilon) \tilde{x}$ 
7       Compute value function:  $V^{(i)} \leftarrow D_{\omega}(x) - D_{\omega}(\tilde{x}) - \lambda(\|\nabla_{\hat{x}} D_{\omega}(\hat{x})\|_2 - 1)^2$ 
8     end
9     Compute the update direction for critic:  $\Delta_{\omega} \leftarrow \nabla_{\omega} \frac{1}{m} \sum_{i=1}^m V^{(i)}$ 
10    Update critic parameters:  $\omega \leftarrow \omega + \text{Adam}(\omega, \Delta_{\omega}, \alpha, \beta_1, \beta_2)$ 
11  end
12  Sample a batch of latent variables  $\{z^{(i)}\}_{i=1}^m \sim p(z)$ .
13  Compute the update direction for generator:  $\Delta_{\theta} \leftarrow \nabla_{\theta} \frac{1}{m} \sum_{i=1}^m D_{\omega}(G_{\theta}(z^{(i)}))$ 
14  Update generator's parameters:  $\theta \leftarrow \theta + \text{Adam}(\theta, \Delta_{\theta}, \alpha, \beta_1, \beta_2)$ 
15 end

```

19.6 Wasserstein GAN with Lipschitz Penalty (WGAN-LP)

19.6.1 Introduction

本节选自 On the regularization of Wasserstein GANs[?]

19.6.2 Model formulation

Penalizing the violation of the Lipschitz constraint

$$\left(\max \left\{ 0, \frac{|f(g(\mathbf{x})) - f(\mathbf{y})|}{\|g(\mathbf{x}) - \mathbf{y}\|_2} - 1 \right\} \right)^2$$

其中 $f(\cdot) : \mathbb{R}^{\mathcal{Y}} \rightarrow \mathbb{R}$ 为判别器对应的判别函数； $g(\cdot) : \mathbb{R}^{\mathcal{X}} \rightarrow \mathbb{R}^{\mathcal{Y}}$ 为生成器对应的生成函数。

注意：区分 WGAN-LP[?] 和 Mode-Seeking GAN[?]

19.7 Wasserstein GAN with Wasserstein Gradient Regularization (WWGAN)

19.7.1 Introduction

本节选自Wasserstein of Wasserstein Loss for Learning Generative Models[?]

19.8 Wasserstein GAN with Consistency Term (CT-GAN) (未完成)

19.8.1 Introduction

本文选自Improving the Improved Training of Wasserstein GANs: A Consistency Term and Its Dual Effect[?]

19.9 Virtual Adversarial Lipschitz Regularization (未完成)

19.9.1 Introduction

本节选自Virtual Adversarial Lipschitz Regularization[?]

19.9.2 model formulation

19.10 Survey and Summary (未完成)

19.10.1 Introduction

本节对一系列的 GAN 进行总结归纳，其中内容选自综述论文：

- Generative Adversarial Networks: A Survey and Taxonomy

本章将从模型构造，目标函数设计以及实际应用场景三个方面介绍不同的 GANs 模型

19.10.2 Architecture-variant GANs

19.10.3 Loss-variant GANs

19.10.4 Applications of GANs

- Fingerprint Producer:
 - Fingerprint Inpainting with Generative Models
 - DeepMasterPrints: Generating MasterPrints for Dictionary Attacks via Latent Variable Evolution[?]
 - Finger-GAN: Generating Realistic Fingerprint Images Using Connectivity Imposed GAN[?]
- Semantic image inpainting:
 - Semantic Image Inpainting with Deep Generative Models[?]
- Text to Image Synthesis:
 - Generative Adversarial Text to Image Synthesis[?]

19.10.5 Personal ideas of GANs

About Fingerprints-GANs 图为DeepMasterPrints: Generating MasterPrints for Dictionary Attacks via Latent Variable Evolution[?] 该文提供了几条关于指纹合成很关键的**指导思想**：

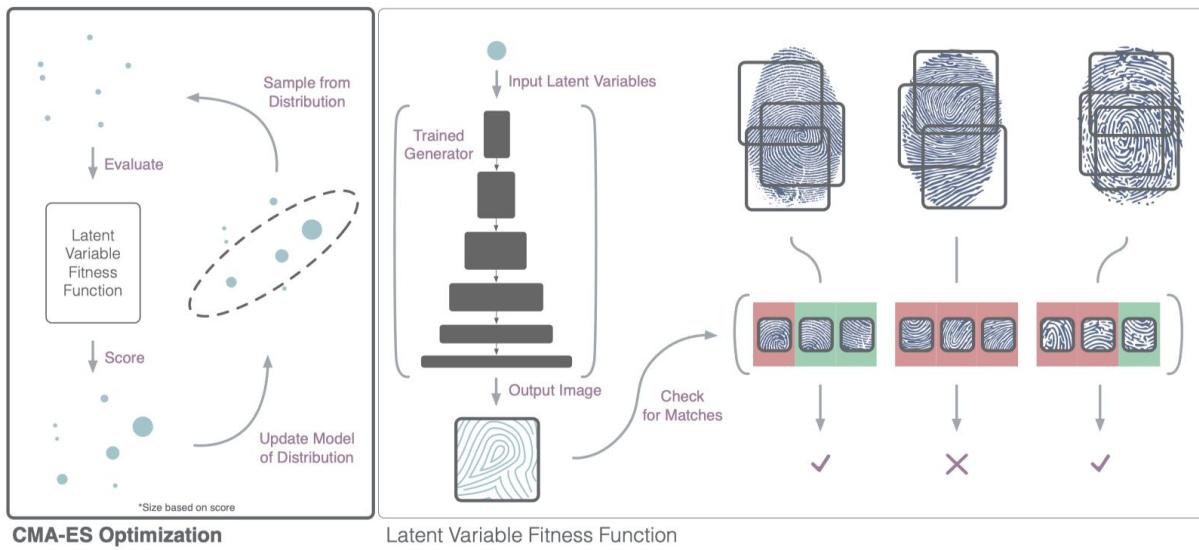


图 101: Latent Variable Evolution with a trained network. On the left is a high level overview of CMA-ES and the box on the right shows how the latent variables are evaluated.

- 在一些实际设备（譬如智能手机、指纹传感器等）的设计中，由于人体工程学的原因，传感器只会获取用户指纹的一部分，而部分指纹又不像全部指纹那样特殊，因此一个部分指纹和另一个部分指纹被错误匹配的概率就大大增强，于是这就有了“可乘之机”
- 使用标准的 GANs 方法（不加额外约束）是远远不够的，因为迄今为止的 GAN 模型都只是识别了 domain 的视觉风格（visual realism），而没有和许多图的许多局部细节进行匹配

20 Variational Autoencoders (VAEs)

20.1 An Introduction to Variational Autoencoders (未完成)

20.1.1 Introduction

本文选自 [An Introduction to Variational Autoencoders](#)

21 Graph Neural Networks (GNNs)

21.1 Overview of GCNs

In this section, the notations used in this survey are listed for convenience. Besides, some definitions and categorizations about graph neural networks are introduced, and some related surveys are also provided.

The notations used in GCNs show as follows:

表 1: Commonly used notations of graph neural networks

Notations	Description
$ \cdot $	The length of a set.
*	Graph convolution operation.
\odot	Element-wise product.
\mathcal{G}	A graph.
\mathcal{V}	The set of nodes in a graph.
v	A node $v \in \mathcal{V}$.
\mathcal{E}	The set of edges in a graph.
e_{ij}	An edge $e_{ij} \in \mathcal{E}$.
$\mathcal{N}(v)$	The neighbors of a node v .
\mathbf{A}	The graph adjacency matrix.
\mathbf{A}^T	The transpose of the matrix \mathbf{A} .
\mathbf{A}^n	The n^{th} power of \mathbf{A} .
$[\mathbf{A}, \mathbf{B}]$	The concatenation of \mathbf{A} and \mathbf{B} .
\mathbf{D}	The degree matrix corresponding \mathbf{A} . $\mathbf{D}_{ii} = \sum_{j=1}^n \mathbf{A}_{ij}$.
n	The number of nodes, $n = \mathcal{V} $.
m	The number of edges, $m = \mathcal{E} $.
d	The dimension of a node feature vector.
$\mathbf{X} \in \mathbb{R}^{d \times n}$	The feature matrix of a graph (for theoretical derivation).
$\mathbf{X}^T \in \mathbb{R}^{n \times d}$	The feature matrix of a graph (for programming implementation).
$\mathbf{x}^{(i)} \in \mathbb{R}^d$	The feature vector of the node v_i .
$\mathbf{x}_j^{(1:n)} \in \mathbb{R}^n$	The feature vector of a graph in the case $d = j$.

The definition of graph shows as follows:

Definition 21.1 (Graph[?][?]). A graph is represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where \mathcal{V} is the set of vertices or nodes $\mathcal{V} = \{v_1, \dots, v_n\}$, and \mathcal{E} is the set of edges.

Let $v_i \in \mathcal{V}$ to denote a vertice and $e_{ij} = (v_i, v_j) \in \mathcal{E}$ to denote an edge pointing from v_j to v_i .

The neighborhood of a node v is defined as $\mathcal{N}(v) = \{u \in \mathcal{V} | (v, u) \in \mathcal{E}\}$.

The adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a symmetric (typically sparse) matrix where $a_{ij} \in \mathbb{R}$ denotes the edge weight between nodes v_i and v_j . A missing edge is represented through $\mathbf{A}_{ij} = 0, \forall (v_i, v_j) \notin \mathcal{E}$. And $\mathbf{A}_{ii} = 0, \forall i \in \{1, \dots, n\}$.

Also, we define the degree matrix $\mathbf{D} = \text{diag}(d_1, \dots, d_n)$ as a diagonal matrix where each entry on the diagonal is equal to the row-sum of the adjacency matrix: $d_i = \mathbf{D}_{ii} = \sum_{j=1}^n \mathbf{A}_{ij}$.

Each node v_i in the graph has a corresponding d -dimensional feature vector $\mathbf{x}^{(i)} \in \mathbb{R}^d$. The entire feature matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$ stacks n nodes feature vectors $[\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(i)}, \dots, \mathbf{x}^{(n)}]$. Each node belongs to one out of C classes and can be labeled with a C -dimensional one-hot vector $\mathbf{y}^{(i)} \in \{0, 1\}^C$. We only know the labels of a subset of all nodes \mathcal{V} and want to predict the unknown labels.

In summary, some notations of graph constants defined as follows:

- $\mathbf{A} \in \mathbb{R}^{n \times n}$: Adjacency matrix of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
 - $(v_i, v_j) \in \mathcal{E} \Leftrightarrow \mathbf{A}_{ij} \neq 0$
 - $(v_i, v_j) \notin \mathcal{E} \Leftrightarrow \mathbf{A}_{ij} = 0$
 - $\mathbf{A}_{ii} = 0, \forall i \in \{1, \dots, n\}$
- $\mathbf{I} \in \mathbb{R}^{n \times n}$: Identity matrix containing identity self-loops
- $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$: Augmented adjacency matrix, namely adjacency matrix with self-loops
 - $\tilde{\mathbf{A}}_{ij} = \mathbf{A}_{ij} + \mathbf{I}_{ij}, \forall i, j \in \{1, \dots, n\}$
 - $\tilde{\mathbf{A}}_{ii} = \mathbf{A}_{ii} + \mathbf{I}_{ii} = \mathbf{A}_{ii} + 1, \forall i \in \{1, \dots, n\}$
 - $\tilde{\mathbf{A}}_{ij} = \mathbf{A}_{ij} + \mathbf{I}_{ij} = \mathbf{A}_{ij}, \forall i, j \in \{1, \dots, n\}, i \neq j$
- $\mathbf{D} = \text{diag}(d_1, \dots, d_n)$: Degree matrix corresponding \mathbf{A}
 - $d_i = \mathbf{D}_{ii} = \sum_{j=1}^n \mathbf{A}_{ij}, \forall i \in \{1, \dots, n\}$
- $\tilde{\mathbf{D}} = \text{diag}(\tilde{d}_1, \dots, \tilde{d}_n)$: Augmented degree matrix corresponding $\tilde{\mathbf{A}}$
 - $\tilde{d}_i = \tilde{\mathbf{D}}_{ii} = \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} = \sum_{j=1}^n (\mathbf{A}_{ij} + \mathbf{I}_{ij})$

Comprehension of local smoothing and normalized Laplacian. Here we consider four cases to introduce normalized Laplacian. Summary of four cases show as follows:

表 2: Local smoothing using asymmetric or symmetric normalized edge weight

Normalized Weight	Element Operation	Matrix Operation
$\frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}}$	$\bar{\mathbf{x}}^{(i)} = \tilde{\mathbf{D}}_{ii}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} \mathbf{x}^{(j)}$	$\bar{\mathbf{X}}^T = \tilde{\mathbf{D}}^{-1} \tilde{\mathbf{A}} \mathbf{X}^T$
$\frac{\tilde{\mathbf{A}}_{ij}}{(\sum_{k=1}^n \tilde{\mathbf{A}}_{ik})^{\frac{1}{2}} (\sum_{l=1}^n \tilde{\mathbf{A}}_{jl})^{\frac{1}{2}}}$	$\bar{\mathbf{x}}^{(i)} = \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} \mathbf{x}^{(j)}$	$\bar{\mathbf{X}}^T = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}^T$

表 3: Laplacian using asymmetric or symmetric normalized edge weight

Normalized Weight	Element Operation	Matrix Operation
$\frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}}$	$\mathbf{x}^{(i)} - \bar{\mathbf{x}}^{(i)} = \sum_{j=1}^n \tilde{\mathbf{D}}_{ii}^{-1} \tilde{\mathbf{A}}_{ij} (\mathbf{x}^{(i)} - \mathbf{x}^{(j)})$	$\mathbf{X}^T - \bar{\mathbf{X}}^T = \left(\mathbf{I} - \tilde{\mathbf{D}}^{-1} (\mathbf{A} + \mathbf{I}) \right) \mathbf{X}^T$
$\frac{\tilde{\mathbf{A}}_{ij}}{(\sum_{k=1}^n \tilde{\mathbf{A}}_{ik})^{\frac{1}{2}} (\sum_{l=1}^n \tilde{\mathbf{A}}_{jl})^{\frac{1}{2}}}$	$\mathbf{x}^{(i)} - \bar{\mathbf{x}}^{(i)} = \mathbf{x}^{(i)} - \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} \mathbf{x}^{(j)}$	$\mathbf{X}^T - \bar{\mathbf{X}}^T = \left(\mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}} (\mathbf{A} + \mathbf{I}) \tilde{\mathbf{D}}^{-\frac{1}{2}} \right) \mathbf{X}^T$

Local smoothing with asymmetric normalized argumented adjacency Consider to aggregate neighbors' feature vector:

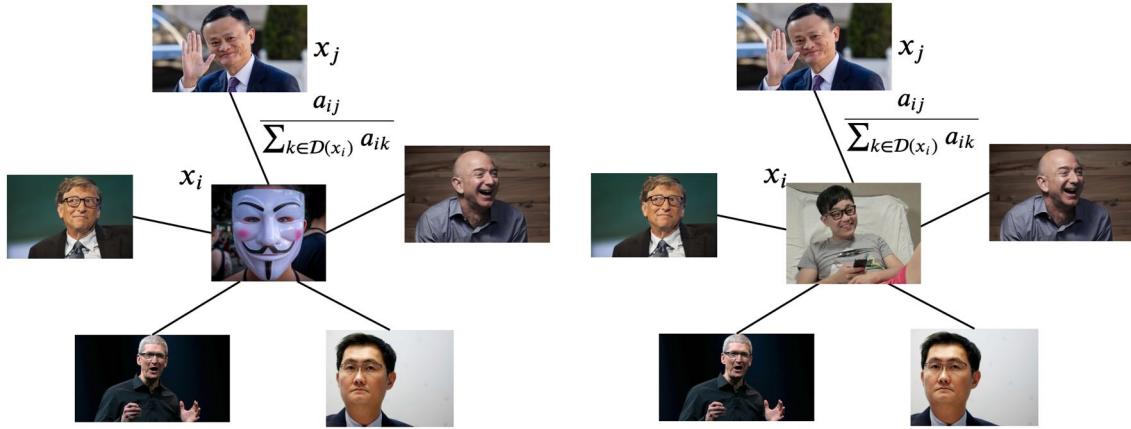


图 102: Social network examples for predicting annual salary informations. Using argumented adjacency matrix $\tilde{\mathbf{A}}$ to replace original adjacency matrix \mathbf{A} .

$$\begin{aligned}
 \bar{\mathbf{x}}^{(i)} &= \sum_{j=1}^n \frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}} \mathbf{x}^{(j)} \quad (\text{if } \frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}} \in \mathbb{R} \text{ is the normalized argumented edge weight between } v_i \text{ and } v_j, \text{ centring with } v_i) \\
 &= \sum_{j=1}^n \frac{\tilde{\mathbf{A}}_{ij}}{\tilde{\mathbf{D}}_{ii}} \mathbf{x}^{(j)} \quad (\tilde{\mathbf{D}} \text{ is the degree matrix corresponding } \tilde{\mathbf{A}}) \\
 &= \sum_{j=1}^n \tilde{\mathbf{D}}_{ii}^{-1} \tilde{\mathbf{A}}_{ij} \mathbf{x}^{(j)} \\
 &= \tilde{\mathbf{D}}_{ii}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} \mathbf{x}^{(j)} \in \mathbb{R}^d
 \end{aligned}$$

where $\bar{\cdot}$ means an aggregator function. Corresponding matrix operation expression (in programming implementation) as follows:

$$\begin{aligned}
 \bar{\mathbf{X}}^T &= \left[\bar{\mathbf{x}}^{(1)}, \dots, \bar{\mathbf{x}}^{(i)}, \dots, \bar{\mathbf{x}}^{(n)} \right]^T \\
 &= \begin{bmatrix} (\bar{\mathbf{x}}^{(1)})^T \\ \vdots \\ (\bar{\mathbf{x}}^{(i)})^T \\ \vdots \\ (\bar{\mathbf{x}}^{(n)})^T \end{bmatrix}
 \end{aligned}$$

$$\begin{aligned}
& \left[\tilde{\mathbf{D}}_{11}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{1j} (\mathbf{x}^{(j)})^T \right] \\
& \vdots \\
& = \left[\tilde{\mathbf{D}}_{ii}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} (\mathbf{x}^{(j)})^T \right] \\
& \vdots \\
& \left[\tilde{\mathbf{D}}_{nn}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{nj} (\mathbf{x}^{(j)})^T \right] \\
& = \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-1} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-1} \end{bmatrix} \begin{bmatrix} \sum_{j=1}^n \tilde{\mathbf{A}}_{1j} (\mathbf{x}^{(j)})^T \\ \vdots \\ \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} (\mathbf{x}^{(j)})^T \\ \vdots \\ \sum_{j=1}^n \tilde{\mathbf{A}}_{nj} (\mathbf{x}^{(j)})^T \end{bmatrix} \\
& = \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-1} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-1} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{A}}_{11} & \cdots & \tilde{\mathbf{A}}_{1j} & \cdots & \tilde{\mathbf{A}}_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{\mathbf{A}}_{i1} & \cdots & \tilde{\mathbf{A}}_{ij} & \cdots & \tilde{\mathbf{A}}_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{\mathbf{A}}_{n1} & \cdots & \tilde{\mathbf{A}}_{nj} & \cdots & \tilde{\mathbf{A}}_{nn} \end{bmatrix} \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ \vdots \\ (\mathbf{x}^{(j)})^T \\ \vdots \\ (\mathbf{x}^{(n)})^T \end{bmatrix} \\
& = \tilde{\mathbf{D}}^{-1} \tilde{\mathbf{A}} \mathbf{X}^T
\end{aligned}$$

Local smoothing with symmetric normalized argumented adjacency Above we use asymmetric normalized edge weight $\frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}}$, however, for example, some people v_j are friends with everyone, even if v_i and v_j are very close, when these persons v_j are green tea bitches, we need to limit the role of these persons v_j , therefore consider symmetric nomalized edge weight $\frac{\tilde{\mathbf{A}}_{ij}}{(\sum_{k=1}^n \tilde{\mathbf{A}}_{ik})^{\frac{1}{2}} (\sum_{l=1}^n \tilde{\mathbf{A}}_{jl})^{\frac{1}{2}}}$:

$$\begin{aligned}
\bar{\mathbf{x}}^{(i)} &= \sum_{j=1}^n \frac{\tilde{\mathbf{A}}_{ij}}{\left(\sum_{k=1}^n \tilde{\mathbf{A}}_{ik} \right)^{\frac{1}{2}} \left(\sum_{l=1}^n \tilde{\mathbf{A}}_{jl} \right)^{\frac{1}{2}}} \mathbf{x}^{(j)} \\
&= \sum_{j=1}^n \frac{\tilde{\mathbf{A}}_{ij}}{\tilde{\mathbf{D}}_{ii}^{\frac{1}{2}} \tilde{\mathbf{D}}_{jj}^{\frac{1}{2}}} \mathbf{x}^{(j)} \\
&= \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} \mathbf{x}^{(j)}
\end{aligned}$$

where $\bar{\cdot}$ means an aggregator function. Corresponding matrix operation expression (in programming implementation) as follows:

$$\bar{\mathbf{X}}^T = \left[\bar{\mathbf{x}}^{(1)}, \dots, \bar{\mathbf{x}}^{(i)}, \dots, \bar{\mathbf{x}}^{(n)} \right]^T$$

$$\begin{aligned}
&= \begin{bmatrix} (\bar{\mathbf{x}}^{(1)})^T \\ \vdots \\ (\bar{\mathbf{x}}^{(i)})^T \\ \vdots \\ (\bar{\mathbf{x}}^{(n)})^T \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} \sum_{j=1}^n \tilde{\mathbf{A}}_{1j} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} (\mathbf{x}^{(j)})^T \\ \vdots \\ \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} (\mathbf{x}^{(j)})^T \\ \vdots \\ \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \sum_{j=1}^n \tilde{\mathbf{A}}_{nj} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} (\mathbf{x}^{(j)})^T \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{A}}_{11} & \cdots & \tilde{\mathbf{A}}_{1j} & \cdots & \tilde{\mathbf{A}}_{1n} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \tilde{\mathbf{A}}_{i1} & \cdots & \tilde{\mathbf{A}}_{ij} & \cdots & \tilde{\mathbf{A}}_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{\mathbf{A}}_{n1} & \cdots & \tilde{\mathbf{A}}_{nj} & \cdots & \tilde{\mathbf{A}}_{nn} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \end{bmatrix} \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ \vdots \\ (\mathbf{x}^{(j)})^T \\ \vdots \\ (\mathbf{x}^{(n)})^T \end{bmatrix} \\
&= \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}^T
\end{aligned}$$

Normalization Laplacian with asymmetric argumented adjacency Above we only consider the normalization for adjacency matrix for computing the estimation of feature vector using local smoothing (graph smoothing), if we consider Laplacian matrix for computing the residual between feature vector and its corresponding local smoothing feature vector, we have:

$$\begin{aligned}
\mathbf{x}^{(i)} - \bar{\mathbf{x}}^{(i)} &= \mathbf{x}^{(i)} - \sum_{j=1}^n \frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}} \mathbf{x}^{(j)} \\
&= \mathbf{x}^{(i)} \sum_{j=1}^n \frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}} - \sum_{j=1}^n \frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}} \mathbf{x}^{(j)} \quad (\because \sum_{j=1}^n \frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}} = 1) \\
&= \sum_{j=1}^n \frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}} (\mathbf{x}^{(i)} - \mathbf{x}^{(j)}) \\
&= \sum_{j=1}^n \frac{\tilde{\mathbf{A}}_{ij}}{\tilde{\mathbf{D}}_{ii}} (\mathbf{x}^{(i)} - \mathbf{x}^{(j)})
\end{aligned}$$

Corresponding matrix operation expression (in programming implementation) as follows:

$$\begin{aligned}
\mathbf{X}^T - \bar{\mathbf{X}}^T &= \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ \vdots \\ (\mathbf{x}^{(i)})^T \\ \vdots \\ (\mathbf{x}^{(n)})^T \end{bmatrix} - \begin{bmatrix} (\bar{\mathbf{x}}^{(1)})^T \\ \vdots \\ (\bar{\mathbf{x}}^{(i)})^T \\ \vdots \\ (\bar{\mathbf{x}}^{(n)})^T \end{bmatrix} \\
&= \begin{bmatrix} (\mathbf{x}^{(1)})^T \tilde{\mathbf{D}}_{11}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{1j} \\ \vdots \\ (\mathbf{x}^{(i)})^T \tilde{\mathbf{D}}_{ii}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} \\ \vdots \\ (\mathbf{x}^{(n)})^T \tilde{\mathbf{D}}_{nn}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{nj} \end{bmatrix} - \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{1j} (\mathbf{x}^{(j)})^T \\ \vdots \\ \tilde{\mathbf{D}}_{ii}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} (\mathbf{x}^{(j)})^T \\ \vdots \\ \tilde{\mathbf{D}}_{nn}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{nj} (\mathbf{x}^{(j)})^T \end{bmatrix} \\
&= \begin{bmatrix} (\mathbf{x}^{(1)})^T \tilde{\mathbf{D}}_{11}^{-1} \tilde{\mathbf{D}}_{11} \\ \vdots \\ (\mathbf{x}^{(i)})^T \tilde{\mathbf{D}}_{ii}^{-1} \tilde{\mathbf{D}}_{ii} \\ \vdots \\ (\mathbf{x}^{(n)})^T \tilde{\mathbf{D}}_{nn}^{-1} \tilde{\mathbf{D}}_{nn} \end{bmatrix} - \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{1j} (\mathbf{x}^{(j)})^T \\ \vdots \\ \tilde{\mathbf{D}}_{ii}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} (\mathbf{x}^{(j)})^T \\ \vdots \\ \tilde{\mathbf{D}}_{nn}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{nj} (\mathbf{x}^{(j)})^T \end{bmatrix} \quad (\because \tilde{\mathbf{D}}_{ii}^{-1} = \sum_{j=1}^n \tilde{\mathbf{A}}_{ij}) \\
&= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-1} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-1} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{D}}_{11} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn} \end{bmatrix} \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ \vdots \\ (\mathbf{x}^{(i)})^T \\ \vdots \\ (\mathbf{x}^{(n)})^T \end{bmatrix} - \tilde{\mathbf{D}}^{-1} \tilde{\mathbf{A}} \mathbf{X}^T \\
&= \tilde{\mathbf{D}}^{-1} \tilde{\mathbf{D}} \mathbf{X}^T - \tilde{\mathbf{D}}^{-1} \tilde{\mathbf{A}} \mathbf{X}^T \\
&= \tilde{\mathbf{D}}^{-1} (\tilde{\mathbf{D}} - \tilde{\mathbf{A}}) \mathbf{X}^T = \tilde{\mathbf{D}}^{-1} \tilde{\mathbf{L}} \mathbf{X}^T
\end{aligned}$$

Normalization Laplacian with symmetric argumented adjacency Similarly, consider asymmetric case:

$$\begin{aligned}
\mathbf{x}^{(i)} - \bar{\mathbf{x}}^{(i)} &= \mathbf{x}^{(i)} - \sum_{j=1}^n \frac{\tilde{\mathbf{A}}_{ij}}{\left(\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}\right)^{\frac{1}{2}} \left(\sum_{l=1}^n \tilde{\mathbf{A}}_{jl}\right)^{\frac{1}{2}}} \mathbf{x}^{(j)} \\
&= \mathbf{x}^{(i)} - \sum_{j=1}^n \frac{\tilde{\mathbf{A}}_{ij}}{\tilde{\mathbf{D}}_{ii}^{\frac{1}{2}} \tilde{\mathbf{D}}_{jj}^{\frac{1}{2}}} \mathbf{x}^{(j)} \\
&= \mathbf{x}^{(i)} - \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} \mathbf{x}^{(j)}
\end{aligned}$$

Corresponding matrix operation expression (in programming implementation) as follows:

$$\begin{aligned}
\mathbf{X}^T - \bar{\mathbf{X}}^T &= \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ \vdots \\ (\mathbf{x}^{(i)})^T \\ \vdots \\ (\mathbf{x}^{(n)})^T \end{bmatrix} - \begin{bmatrix} (\bar{\mathbf{x}}^{(1)})^T \\ \vdots \\ (\bar{\mathbf{x}}^{(i)})^T \\ \vdots \\ (\bar{\mathbf{x}}^{(n)})^T \end{bmatrix} \\
&= \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ \vdots \\ (\mathbf{x}^{(i)})^T \\ \vdots \\ (\mathbf{x}^{(n)})^T \end{bmatrix} - \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} \sum_{j=1}^n \tilde{\mathbf{A}}_{1j} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} (\mathbf{x}^{(j)})^T \\ \vdots \\ \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} (\mathbf{x}^{(j)})^T \\ \vdots \\ \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \sum_{j=1}^n \tilde{\mathbf{A}}_{nj} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} (\mathbf{x}^{(j)})^T \end{bmatrix} \\
&= \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{D}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}^T - \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}^T \quad (\because \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{D}} \tilde{\mathbf{D}}^{-\frac{1}{2}} = \mathbf{I}) \\
&= (\mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}) \mathbf{X}^T \\
&= (\mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}} (\mathbf{A} + \mathbf{I}) \tilde{\mathbf{D}}^{-\frac{1}{2}}) \mathbf{X}^T
\end{aligned}$$

Lemma 1 (Non-negativity of \mathbf{S}). The augmented normalized Laplacian matrix \mathbf{S} is a symmetric positive semidefinite matrix.

Proof. For any signal $\mathbf{x}_j^{(1:n)} \in \mathbb{R}^n$, it satisfies

$$\begin{aligned}
\because \mathbf{S} \mathbf{x}_j^{(1:n)} &= (\mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}) \mathbf{x}_j^{(1:n)} \\
&= \tilde{\mathbf{D}}^{-\frac{1}{2}} (\tilde{\mathbf{D}} - \tilde{\mathbf{A}}) \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{x}_j^{(1:n)} \\
\because \tilde{\mathbf{D}}^{-\frac{1}{2}} (\tilde{\mathbf{D}} - \tilde{\mathbf{A}}) &= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{D}}_{11} - \tilde{\mathbf{A}}_{11} & \cdots & -\tilde{\mathbf{A}}_{1i} & \cdots & -\tilde{\mathbf{A}}_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ -\tilde{\mathbf{A}}_{i1} & \cdots & \tilde{\mathbf{D}}_{ii} - \tilde{\mathbf{A}}_{ii} & \cdots & -\tilde{\mathbf{A}}_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -\tilde{\mathbf{A}}_{n1} & \cdots & -\tilde{\mathbf{A}}_{ni} & \cdots & \tilde{\mathbf{D}}_{nn} - \tilde{\mathbf{A}}_{nn} \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{11} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{ii} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{nn} \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
\because \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{x}_j^{(1:n)} &= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \end{bmatrix} \begin{bmatrix} x_j^{(1)} \\ x_j^{(i)} \\ \vdots \\ x_j^{(n)} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} x_j^{(1)} \\ \vdots \\ \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} x_j^{(i)} \\ \vdots \\ \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} x_j^{(n)} \end{bmatrix} \\
\therefore \mathbf{S} \mathbf{x}_j^{(1:n)} &= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{11} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{ii} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{nn} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} x_j^{(1)} \\ \vdots \\ \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} x_j^{(i)} \\ \vdots \\ \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} x_j^{(n)} \end{bmatrix} \\
&= \begin{bmatrix} \left(\tilde{\mathbf{D}}_{11}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{11}\right) \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} x_j^{(1)} \\ \vdots \\ \left(\tilde{\mathbf{D}}_{ii}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{ii}\right) \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} x_j^{(i)} \\ \vdots \\ \left(\tilde{\mathbf{D}}_{nn}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{nn}\right) \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} x_j^{(n)} \end{bmatrix} \\
&= \begin{bmatrix} \left(1 - \tilde{\mathbf{D}}_{11}^{-1} \tilde{\mathbf{A}}_{11}\right) x_j^{(1)} \\ \vdots \\ \left(1 - \tilde{\mathbf{D}}_{ii}^{-1} \tilde{\mathbf{A}}_{ii}\right) x_j^{(i)} \\ \vdots \\ \left(1 - \tilde{\mathbf{D}}_{nn}^{-1} \tilde{\mathbf{A}}_{nn}\right) x_j^{(n)} \end{bmatrix}
\end{aligned}$$

The quadratic form associated with \mathbf{S} is

$$\begin{aligned}
(\mathbf{x}_j^{(1:n)})^T \mathbf{S} \mathbf{x}_j^{(1:n)} &= \begin{bmatrix} x_j^{(1)} & \cdots & x_j^{(i)} & \cdots & x_j^{(n)} \end{bmatrix} \begin{bmatrix} \left(1 - \tilde{\mathbf{D}}_{11}^{-1} \tilde{\mathbf{A}}_{11}\right) x_j^{(1)} \\ \vdots \\ \left(1 - \tilde{\mathbf{D}}_{ii}^{-1} \tilde{\mathbf{A}}_{ii}\right) x_j^{(i)} \\ \vdots \\ \left(1 - \tilde{\mathbf{D}}_{nn}^{-1} \tilde{\mathbf{A}}_{nn}\right) x_j^{(n)} \end{bmatrix} \\
&= \sum_{i=1}^n \left(1 - \tilde{\mathbf{D}}_{ii}^{-1} \tilde{\mathbf{A}}_{ii}\right) \left(x_j^{(i)}\right)^2 \\
\because 1 - \tilde{\mathbf{D}}_{ii}^{-1} \tilde{\mathbf{A}}_{ii} &= 1 - \frac{\tilde{\mathbf{A}}_{ii}}{\tilde{\mathbf{D}}_{ii}} \geq 0, \forall i \in \{1, \dots, n\} \\
\therefore (\mathbf{x}_j^{(1:n)})^T \mathbf{S} \mathbf{x}_j^{(1:n)} &\geq 0
\end{aligned}$$

Therefore the augmented normalized Laplacian matrix \mathbf{S} is a symmetric positive semidefinite matrix.

21.1.1 Spectral-based GCNs

Background. Spectral-based methods have a solid mathematical foundation in graph signal processing. They assume graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to be undirected. The symmetric augmented normalized graph Laplacian matrix is a mathematical representation of an undirected graph, defined as

$$\begin{aligned}\mathbf{S} &= \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{L}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \\ &= \tilde{\mathbf{D}}^{-\frac{1}{2}} (\tilde{\mathbf{D}} - \tilde{\mathbf{A}}) \tilde{\mathbf{D}}^{-\frac{1}{2}} \quad (\tilde{\mathbf{L}} = \tilde{\mathbf{D}} - \tilde{\mathbf{A}} \text{ is the Laplacian matrix corresponding to the Augmented Adjacency matrix } \tilde{\mathbf{A}}) \\ &= \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{D}} \tilde{\mathbf{D}}^{-\frac{1}{2}} - \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \\ &= \mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \\ &= \mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}} (\mathbf{A} + \mathbf{I}) \tilde{\mathbf{D}}^{-\frac{1}{2}}\end{aligned}$$

where:

- $\mathbf{A} \in \mathbb{R}^{n \times n}$: Adjacency matrix of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
 - $(v_i, v_j) \in \mathcal{E} \Leftrightarrow \mathbf{A}_{ij} \neq 0$
 - $(v_i, v_j) \notin \mathcal{E} \Leftrightarrow \mathbf{A}_{ij} = 0$
 - $\mathbf{A}_{ii} = 0, \forall i \in \{1, \dots, n\}$
- $\mathbf{I} \in \mathbb{R}^{n \times n}$: Identity matrix containing identity self-loops
- $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$: Augmented adjacency matrix, namely adjacency matrix with self-loops
 - $\tilde{\mathbf{A}}_{ij} = \mathbf{A}_{ij} + \mathbf{I}_{ij}, \forall i, j \in \{1, \dots, n\}$
 - $\tilde{\mathbf{A}}_{ii} = \mathbf{A}_{ii} + \mathbf{I}_{ii} = \mathbf{A}_{ii} + 1, \forall i \in \{1, \dots, n\}$
 - $\tilde{\mathbf{A}}_{ij} = \mathbf{A}_{ij} + \mathbf{I}_{ij} = \mathbf{A}_{ij}, \forall i, j \in \{1, \dots, n\}, i \neq j$
- $\mathbf{D} = \text{diag}(d_1, \dots, d_n)$: Degree matrix corresponding to \mathbf{A}
 - $d_i = \mathbf{D}_{ii} = \sum_{j=1}^n \mathbf{A}_{ij}, \forall i \in \{1, \dots, n\}$
- $\tilde{\mathbf{D}} = \text{diag}(\tilde{d}_1, \dots, \tilde{d}_n)$: Augmented degree matrix corresponding to $\tilde{\mathbf{A}}$
 - $\tilde{d}_i = \tilde{\mathbf{D}}_{ii} = \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} = \sum_{j=1}^n (\mathbf{A}_{ij} + \mathbf{I}_{ij})$

The normalized graph Laplacian matrix possesses the property of being real symmetric positive semidefinite. With this property, the normalized Laplacian matrix can be factored (eigendecomposition) as:

$$\mathbf{S} = \mathbf{U} \mathbf{U}^T$$

where $\mathbf{U} = [\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_n] \in \mathbb{R}^{n \times n}$ is the matrix of eigenvectors ordered by eigenvalues and $\mathbf{D} = \text{diag}(\lambda_1, \dots, \lambda_n)$ is the diagonal matrix of eigenvalues (spectrum), $\mathbf{D}_{ii} = \lambda_i$.

The eigenvectors of the normalized Laplacian matrix form an orthonormal space, in mathematical words $\mathbf{U}^T \mathbf{U} = \mathbf{I}$. In graph signal processing, a graph signal $\mathbf{x}_j^{(1:n)} \in \mathbb{R}^n$ (value of all nodes in a specific feature dimension) is a feature vector of all nodes of a graph, where $x_j^{(i)}$ is the value of j^{th} dimension of the i^{th} node.

The graph Fourier transform to a signal $\mathbf{x}_j^{(1:n)} \in \mathbb{R}^n$ is defined as $\mathfrak{F}(\mathbf{x}_j^{(1:n)}) = \mathbf{U}^T \mathbf{x}_j^{(1:n)}$ and the inverse graph Fourier transform is defined as $\mathfrak{F}^{-1}(\hat{\mathbf{x}}_j^{(1:n)}) = \mathbf{U} \hat{\mathbf{x}}_j^{(1:n)}$, where $\hat{\mathbf{x}}_j^{(1:n)}$ represents the resulted signal from the graph Fourier transform.

The graph Fourier transform projects the input graph signal to the orthonormal space where the basis is formed by eigenvectors of the normalized graph Laplacian \mathbf{S} . Elements of the transformed signal $\hat{\mathbf{x}}$ are the coordinates of the graph signal in the new space so that the input signal can be represented as $\mathbf{x} = \sum_{i=1}^n \hat{x}_i \mathbf{u}_i$, which is exactly the inverse graph Fourier transform. Now the graph convolution of the input signal \mathbf{x} with a filter $\mathbf{g} \in \mathbb{R}^n$ is defined as

$$\begin{aligned}\mathbf{x} * \mathbf{g} &= \mathfrak{F}^{-1}(\mathfrak{F}(\mathbf{x}) \odot \mathfrak{F}(\mathbf{g})) \\ &= \mathbf{U} (\mathbf{U}^T \mathbf{x} \odot \mathbf{U}^T \mathbf{g})\end{aligned}$$

where \odot denotes the element-wise product and $*$ is a custom notation for graph convolution operation. If we denote a filter as $\mathbf{g}_\theta = \text{diag}(\mathbf{U}^T \mathbf{g}) \in \mathbb{R}^{n \times n}$, then the spectral graph convolution is simplified as

$$\mathbf{x} * \mathbf{g}_\theta = \mathbf{U} \mathbf{g}_\theta \mathbf{U}^T \mathbf{x}$$

Spectral-based GCNs all follow this definition. The key difference lies in the choice of the filter \mathbf{g}_θ .

Spectral Convolutional Neural Network (Spectral CNN).

Reading materials.

- QiMai's Home: Laplacian Smoothing and Graph Convolutional Networks
- Simplifying Graph Convolutional Networks

21.1.2 Spatial-based GCNs

Analogous to the convolutional operation of a conventional CNN on an image, spatial-based methods define graph convolutions based on a node's spatial relations. According to paper: [Simplifying Graph Convolutional Networks](#)[?], similar to CNNs or MLPs, GCNs learn a new feature representation $\mathbf{h}_i^{(k)}$ for the feature \mathbf{x}_i of each node v_i over multiple layers, which is subsequently used as input into a linear classifier. We denote as follows:

- $\mathbf{H}^{(k-1)} \in \mathbb{R}^{n \times d'}$: input node representations of all nodes (a matrix).
- $\mathbf{H}^{(k)} \in \mathbb{R}^{n \times d''}$: output node representations of all nodes (a matrix).

Naturally, the initial node representations are just the original input features:

$$\mathbf{H}^{(0)} = \mathbf{X}^T \in \mathbb{R}^{n \times d}$$

which serve as input to the first GCN layer. A K -layer GCN is identical to applying a K -layer MLP to the feature vector \mathbf{x}_i of each node in the graph, except that **the hidden representation of each node is averaged with its neighbors at the beginning of each layer**. In each graph convolution layer, node representations are updated in three stages:

1. Feature propagation
2. Feature (Linear) transformation
3. Pointwise nonlinear activation

For the sake of clarity, we describe each step in detail.

Feature propagation Feature propagation is what distinguishes a GCN from an MLP. At the beginning of each layer the features \mathbf{h}_i of each node v_i are averaged with the feature vectors in its local neighborhood (**local smoothing**),

$$\bar{\mathbf{h}}_i^{(k)} \leftarrow \frac{1}{d_i + 1} \mathbf{h}_i^{(k-1)} + \sum_{j=1}^n \frac{a_{ij}}{\sqrt{(d_i + 1)(d_j + 1)}} \mathbf{h}_j^{(k-1)}$$

More compactly, we can express this update over the entire graph as a simple matrix operation. Let $\mathbf{S} \in \mathbb{R}^{n \times n}$ denote the symmetric normalized adjacency matrix with added self-loops,

$$\mathbf{S} = \mathbf{D}^{-\frac{1}{2}} \tilde{\mathbf{A}} \mathbf{D}^{-\frac{1}{2}}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and \mathbf{D} is the degree matrix corresponding $\tilde{\mathbf{A}}$. The simultaneous update in above update equation for all nodes becomes a simple sparse matrix multiplication

$$\bar{\mathbf{H}}^{(k)} \leftarrow \mathbf{S} \mathbf{H}^{(k-1)}.$$

Intuitively, **this step smoothes the hidden representations locally along the edges of the graph and ultimately encourages similar predictions among locally connected nodes**.

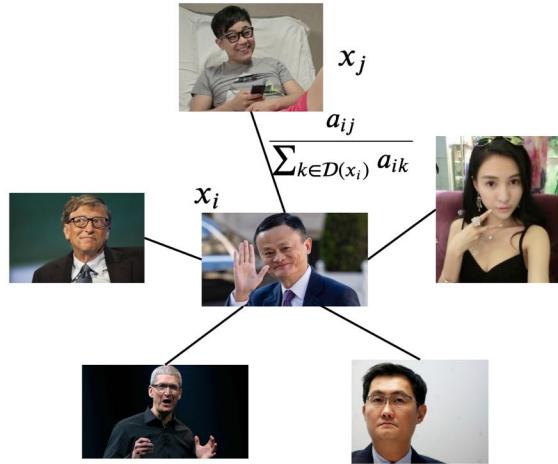


图 103: Social network examples for predicting annual salary informations. Using symmetric normalized edge weight $\frac{\hat{A}_{ij}}{(\sum_{k=1}^n \hat{A}_{ik})^{\frac{1}{2}} (\sum_{l=1}^n \hat{A}_{jl})^{\frac{1}{2}}}$ to replace asymmetric normalized edge weight $\frac{\hat{A}_{ij}}{\sum_{k=1}^n \hat{A}_{ik}}$.

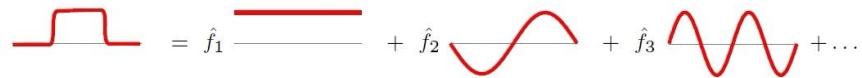


图 104: Fourier transform for a specific function f .

图 105: Schematic layout of a GCN v.s. a SGC. Top row: The GCN transforms the feature vectors repeatedly throughout K layers and then applies a linear classifier on the final representation. Bottom row: the SGC reduces the entire procedure to a simple feature propagation step followed by standard logistic regression. Image source: <https://arxiv.org/pdf/1902.07153.pdf>.

Proof.

$$\begin{aligned}
\bar{\mathbf{H}}^{(k)} &= \begin{bmatrix} \bar{\mathbf{h}}_1^{(k)} \\ \vdots \\ \bar{\mathbf{h}}_i^{(k)} \\ \vdots \\ \bar{\mathbf{h}}_n^{(k)} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{d_1+1} \mathbf{h}_1^{(k-1)} + \sum_{j=1}^n \frac{a_{1j}}{\sqrt{(d_1+1)(d_j+1)}} \mathbf{h}_j^{(k-1)} \\ \vdots \\ \frac{1}{d_i+1} \mathbf{h}_i^{(k-1)} + \sum_{j=1}^n \frac{a_{ij}}{\sqrt{(d_i+1)(d_j+1)}} \mathbf{h}_j^{(k-1)} \\ \vdots \\ \frac{1}{d_n+1} \mathbf{h}_n^{(k-1)} + \sum_{j=1}^n \frac{a_{nj}}{\sqrt{(d_n+1)(d_j+1)}} \mathbf{h}_j^{(k-1)} \end{bmatrix} = \begin{bmatrix} \frac{1}{d_1+1} \mathbf{h}_1^{(k-1)} \\ \vdots \\ \frac{1}{d_i+1} \mathbf{h}_i^{(k-1)} \\ \vdots \\ \frac{1}{d_n+1} \mathbf{h}_n^{(k-1)} \end{bmatrix} + \begin{bmatrix} \sum_{j=1}^n \frac{a_{1j}}{\sqrt{(d_1+1)(d_j+1)}} \mathbf{h}_j^{(k-1)} \\ \vdots \\ \sum_{j=1}^n \frac{a_{ij}}{\sqrt{(d_i+1)(d_j+1)}} \mathbf{h}_j^{(k-1)} \\ \vdots \\ \sum_{j=1}^n \frac{a_{nj}}{\sqrt{(d_n+1)(d_j+1)}} \mathbf{h}_j^{(k-1)} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{d_1+1} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{d_i+1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \frac{1}{d_n+1} \end{bmatrix} \begin{bmatrix} \mathbf{h}_1^{(k-1)} \\ \vdots \\ \mathbf{h}_i^{(k-1)} \\ \vdots \\ \mathbf{h}_n^{(k-1)} \end{bmatrix} + \begin{bmatrix} \sum_{j=1}^n (d_1+1)^{-\frac{1}{2}} a_{1j} (d_j+1)^{-\frac{1}{2}} \mathbf{h}_j^{(k-1)} \\ \vdots \\ \sum_{j=1}^n (d_i+1)^{-\frac{1}{2}} a_{ij} (d_j+1)^{-\frac{1}{2}} \mathbf{h}_j^{(k-1)} \\ \vdots \\ \sum_{j=1}^n (d_n+1)^{-\frac{1}{2}} a_{nj} (d_j+1)^{-\frac{1}{2}} \mathbf{h}_j^{(k-1)} \end{bmatrix} \\
&= \mathbf{D}^{-1} \mathbf{H}^{(k-1)} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \mathbf{H}^{(k-1)} \\
&= \left(\mathbf{D}^{-1} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \right) \mathbf{H}^{(k-1)} \\
&= \left(\mathbf{D}^{-\frac{1}{2}} \mathbf{D}^{-\frac{1}{2}} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \right) \mathbf{H}^{(k-1)} \quad (\because \mathbf{D} \text{ is a diagonal matrix} \therefore \mathbf{D}^{-1} = \mathbf{D}^{-\frac{1}{2}} \mathbf{D}^{-\frac{1}{2}}) \\
&= \left(\mathbf{D}^{-\frac{1}{2}} \mathbf{I} \mathbf{D}^{-\frac{1}{2}} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \right) \mathbf{H}^{(k-1)} \\
&= \mathbf{D}^{-\frac{1}{2}} (\mathbf{A} + \mathbf{I}) \mathbf{D}^{-\frac{1}{2}} \mathbf{H}^{(k-1)} \\
&= \mathbf{D}^{-\frac{1}{2}} \tilde{\mathbf{A}} \mathbf{D}^{-\frac{1}{2}} \mathbf{H}^{(k-1)} \\
&= \mathbf{S} \mathbf{H}^{(k-1)}
\end{aligned}$$

Feature transformation and and nonlinear activation After the local smoothing, a GCN layer is identical to a standard MLP. Each layer is associated with a learned weight (k) and the smoothed hidden feature representations are transformed linearly. Finally, a nonlinear activation function such as ReLU is applied pointwise before outputting feature representation $\mathbf{H}^{(k)}$. In summary, the representation updating rule of the k -th layer is:

$$\mathbf{H}^{(k)} \leftarrow \text{ReLU} \left(\bar{\mathbf{H}}^{(k)} \right)$$

The pointwise nonlinear transformation of the k -th layer is followed by the feature propagation of the $(k+1)$ -th layer. And $\mathbf{H}^{(K)} \in \mathbb{R}^{n \times d_{\text{embed}}}$ is the final embeddings of all nodes.

21.2 DeepWalk: Online Learning of Social Representations (未完成)

21.2.1 Introduction

本节来自论文：DeepWalk: Online Learning of Social Representations[?]

- 目的：学习图（Graph）中每个节点（Vertex）的社交表达（social representations），其中 social representations 实际上是 latent vector representations

21.2.2 Model formulation

Training procedure DeepWalk 算法伪代码如下：

Algorithm 24: DeepWalk(G, ω, d, γ, t)[?]

Input: graph $G(V, E)$;
 window size ω ;
 embedding size d ;
 walks per vertex γ ;
 walk length t .

Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

```

1 Initialization: Sample  $\Phi$  from  $\mathcal{U}^{|V| \times d}$ 
2 Build a binary tree  $T$  from  $V$ 
3 for  $i = 0$  to  $\gamma$  do
4    $\mathcal{O} = \text{Shuffle}(V)$ 
5   for each  $v_i \in \mathcal{O}$  do
6      $\mathcal{W}_{v_i} = \text{RandomWalk}(G, v_i, t)$ 
7     SkipGram( $\Phi, \mathcal{W}_{v_i}, \omega$ )
8   end
9 end

```

Algorithm 25: SkipGram($\Phi, \mathcal{W}_{v_i}, \omega$)

```

for each  $v_j \in \mathcal{W}_{v_i}$  do
  for do
    | for-block
  end
end

```

21.3 LINE: Large-scale Information Network Embedding (未完成)

21.3.1 Introduction

本节来自论文：LINE: Large-scale Information Network Embedding[?]

21.4 Node2vec: Scalable Feature Learning for Networks (未完成)

21.4.1 Introduction

本节来自论文：Node2vec: Scalable Feature Learning for Networks[?]

21.5 SDNE: Structural Deep Network Embedding (未完成)

21.5.1 Introduction

本节来自论文：Structural Deep Network Embedding[?]

21.6 Struc2vec: Learning Node Representations from Structural Identity (未完成)

21.6.1 Introduction

本节来自论文：struc2vec: Learning Node Representations from Structural Identity[?]

21.7 Survey and Summary (未完成)

21.7.1 Introduction

22 Attention Mechanisms

22.1 Self-Attention

22.1.1 Self-attention mechanisms in Computer Vision

在Self-Attention GAN (SAGAN; Zhang et al., 2018)[?] 中

22.1.2 Self-attention mechanisms in Natural Language Processing

在Attention is all you need (Transformer; Vaswani et al., 2017)[?] 中

22.1.3 Reference

- Lil'Log: Attention? Attention![?]

23 Network Optimization and Regularization

23.1 Exponential Moving Average (EMA)

令 t 时刻 EMA 一阶动量为 v_t

23.2 Activation Function

23.2.1 Logistic function (Sigmoid function)

Forwrd-propagation

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Back-propagation

$$\begin{aligned}\sigma'(x) &= -\frac{1}{(1 + e^{-x})^2} \cdot (e^{-x}) \cdot (-1) \\ &= \frac{e^{-x}}{(1 + e^{-x})(1 + e^{-x})} \\ &= \frac{e^{-x}}{1 + e^{-x}} \cdot \frac{1}{1 + e^{-x}} \\ &= \frac{(1 + e^{-x}) - 1}{1 + e^{-x}} \cdot \frac{1}{1 + e^{-x}} \\ &= \left(1 - \frac{1}{1 + e^{-x}}\right) \cdot \frac{1}{1 + e^{-x}} \\ &= (1 - \sigma(x)) \cdot \sigma(x) \quad (\text{i.e. } \sigma(x) = \frac{1}{1 + e^{-x}})\end{aligned}$$

Summary

- Logistic 激活函数可视为一个“挤压”函数，其将实数范围内的输入 $x \in \mathbb{R}$ 挤压到 $(0, 1)$ 空间；
- 当输入 x 在 0 附近时，sigmoid 型函数近似为线性函数；
- 当输入 $x \rightarrow -\infty$ 和 $x \rightarrow +\infty$ 时候，输出越接近 0 和 1。

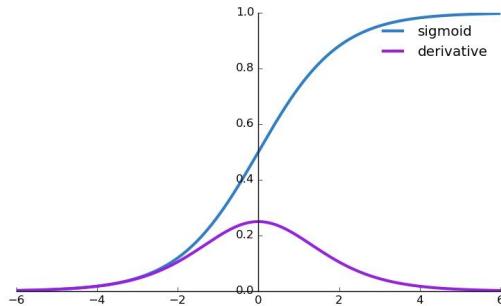


图 106: logistic function and its derivative function. Image source: Ronny Restrepo: Derivative of the Sigmoid function - a worked example

23.2.2 Tanh function

Forwrd-propagation

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, x \in \mathbb{R}$$

Proof. tanh 函数和 sigmoid 函数的关系推导如下：

$$\begin{aligned}\tanh(x) &= \frac{e^x - e^{-x} + (e^x + e^{-x}) - (e^x + e^{-x})}{e^x + e^{-x}} \\ &= \frac{(e^x - e^{-x} + e^x + e^{-x}) - (e^x + e^{-x})}{e^x + e^{-x}} \\ &= \frac{2e^x}{e^x + e^{-x}} - 1 \\ &= \frac{2}{1 + e^{-2x}} - 1 \\ &= 2\sigma(2x) - 1 \quad (\text{i.e. } \sigma(x) = \frac{1}{1 + e^{-x}})\end{aligned}$$

因此可以说 tanh 函数可视为经过放大和平移的 sigmoid 函数，其值域为： $\tanh(x) \in (-1, 1)$ 。

Back-propagation

$$\begin{aligned}\tanh'(x) &= \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} \\ &= \frac{(e^{2x} + 2 + e^{-2x}) - (e^{2x} - 2 + e^{-2x})}{(e^x + e^{-x})^2} \\ &= \frac{4}{(e^x + e^{-x})^2} \\ &= \frac{(e^x + e^{-x})^2 - (e^x + e^{-x})^2 + 4}{(e^x + e^{-x})^2} \\ &= 1 - \frac{(e^x + e^{-x})^2 - 4}{(e^x + e^{-x})^2} \\ &= 1 - \frac{e^{2x} + 2e^x e^{-x} + e^{-2x} - 4}{(e^x + e^{-x})^2} \\ &= 1 - \frac{e^{2x} + e^{-2x} - 2}{(e^x + e^{-x})^2} \\ &= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\ &= 1 - \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2 \\ &= 1 - (\tanh(x))^2\end{aligned}$$

Summary Tanh 函数相对于 Logistic 函数作为激活函数的优势是，Tanh 函数的输出是 0 中心 (zero-centered) 的，而 Logistic 函数的输出恒大于 0。非零中心化的输出会使得其后一层的神经元的输入发生偏置偏移 (bias shift)，并进一步使得梯度下降的收敛速度变慢。

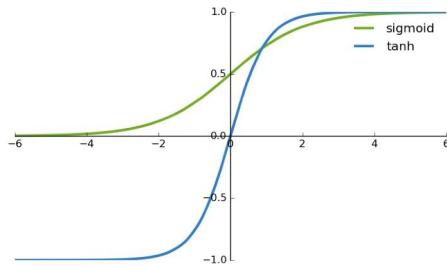


图 108: tanh activation function vs sigmoid activation function. Image source: Ronny Restrepo: Calculating the derivative of Tanh - step by step

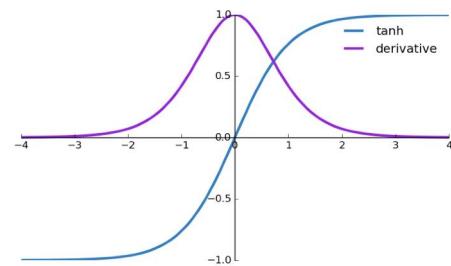


图 109: tanh function and its derivative function. Image source: Ronny Restrepo: Calculating the derivative of Tanh - step by step

23.2.3 ReLU function

论文出处：[Deep Sparse Rectifier Neural Networks](#)^[?] [Deep Learning using Rectified Linear Units \(ReLU\)](#)^[?]。

Forward-propagation

$$\begin{aligned} \text{ReLU}(x) &= \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \\ &= \max(0, x) \end{aligned}$$

Back-propagation

$$\text{ReLU}'(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Summary

- 优点：

1. 拥有生物上的解释性 (e.g. 单侧抑制、宽兴奋边界)。在生物神经网络中，同时处于兴奋状态的神经元非常稀疏。人脑中在同一时刻大概只有 1%~4% 的神经元处于活跃状态。Sigmoid 型激活函数会导致一个非稀疏的神经网络，而 ReLU 却具有很好的稀疏性，大约 50% 的神经元会处于激活状态。
2. 在优化方面，相比于 Sigmoid 型函数的两端饱和，ReLU 函数为左饱和函数，且在 $x > 0$ 时导数为 1，因此在一定程度上缓解了神经网络的梯度消失问题，加速梯度下降的收敛速度。

- 缺点：

1. 输出是非零中心化的，给后一层的神经网络引入偏置偏移，会影响梯度下降的效率
2. ReLU 神经元在训练时比较容易“死亡”。在训练时，如果参数在一次不恰当的更新后，第一个隐藏层中的某个 ReLU 神经元在所有的训练数据上都不能被激活，那么这个神经元自身参数的梯度永远都会是 0，在以后的训练过程中永远不能被激活。这种现象称为死亡 ReLU 问题 (Dying ReLU Problem)，并且也有可能发生在其它隐藏层。因此实际中为了避免上述情况，有几种 ReLU 的变种也会被广泛使用。

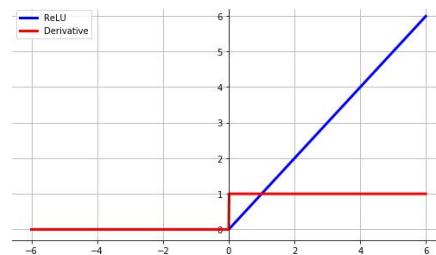


图 109: relu function and its derivative function.

23.2.4 Leaky-ReLU function

论文出处：[Empirical Evaluation of Rectified Activations in Convolution Network\[?\]](#)。

23.2.5 Parametric-ReLU function

论文出处：[Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification\[?\]](#)。

23.3 Network Initialization

23.3.1 Xavier initialization

论文出处：[Understanding the difficulty of training deep feedforward neural networks\[?\]](#)。

Analytics 从正向传播和反向传播两个方向分析单一链路上神经元的均值和方差。

Analytics of forward-propagation 正向传播中，假设第 l 层的某一个（第 j 个）隐层神经元 $z_j^{(l)}$ ，接收前一层（第 $l-1$ 层）共 $n^{(l-1)}$ 个神经元的输出 $a_i^{(l-1)}$, $i \in \{1, \dots, n^{(l-1)}\}$ ，对应线路上的权重为 $w_{i,j}^{(l-1,l)}$, $i \in \{1, \dots, n^{(l-1)}\}$ ，即：

$$z_j^{(l)} = \sum_{i=1}^{n^{(l-1)}} w_{i,j}^{(l-1,l)} a_i^{(l-1)}$$

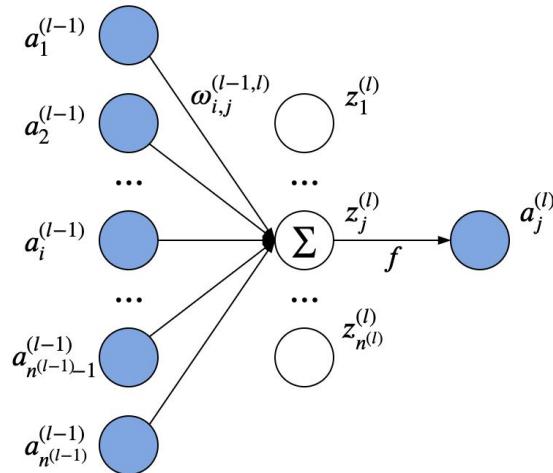


图 110: 正向传播中典型的神经元关系结构

为了避免初始化参数 $w_{i,j}^{(l-1,l)} = \{w_{1,j}^{(l-1,l)}, \dots, w_{i,j}^{(l-1,l)}, \dots, w_{n^{(l-1)},j}^{(l-1,l)}\}$ 使得激活值 $a_j^{(l)} = f(z_j^{(l)})$ 变得饱和，我们需要 **尽量使得 $z_j^{(l)} \in \mathbb{R}$ 处于激活函数的线性区间**，也就是其绝对值 $|z_j^{(l)}|$ 相对较小，此时该神经元的激活函数值满足：

$$a_j^{(l)} = f(z_j^{(l)}) \approx z_j^{(l)}$$

假设在初始化时，参数 $w_{i,j}^{(l-1,l)}$ 和输入 $a_i^{(l-1)}$ 的均值都为 0，并且相互独立，则激活输出 $a_j^{(l)}$ 的均值为：

$$\begin{aligned} \because \mathbb{E}[z_j^{(l)}] &= \mathbb{E}\left[\sum_{i=1}^{n^{(l-1)}} w_{i,j}^{(l-1,l)} a_i^{(l-1)}\right] \\ &= \sum_{i=1}^{n^{(l-1)}} \mathbb{E}[w_{i,j}^{(l-1,l)} a_i^{(l-1)}] \quad (\because w_{i,j}^{(l-1,l)} a_i^{(l-1)}, \forall i \in \{1, \dots, n^{(l-1)}\} \text{ 之间相互独立}) \\ &= \sum_{i=1}^{n^{(l-1)}} \mathbb{E}[w_{i,j}^{(l-1,l)}] \mathbb{E}[a_i^{(l-1)}] \quad (\because w_{i,j}^{(l-1,l)} \text{ 和 } a_i^{(l-1)} \text{ 相互独立}) \end{aligned}$$

$$\begin{aligned}
&= \sum_{i=1}^{n^{(l-1)}} 0 \times 0 = 0 \quad (\because \mathbb{E}[w_{i,j}^{(l-1,l)}] = 0, \mathbb{E}[a_i^{(l-1)}] = 0) \\
\therefore \mathbb{E}[a_j^{(l)}] &\approx \mathbb{E}[z_j^{(l)}] = 0 \quad (\because \text{初始化时 } a_j^{(l)} = f(z_j^{(l)}) \approx z_j^{(l)})
\end{aligned}$$

同时激活输出 $a_j^{(l)}$ 的方差为：

$$\begin{aligned}
\because \text{Var}[z_j^{(l)}] &= \text{Var}\left[\sum_{i=1}^{n^{(l-1)}} w_{i,j}^{(l-1,l)} a_i^{(l-1)}\right] \\
&= \sum_{i=1}^{n^{(l-1)}} \text{Var}[w_{i,j}^{(l-1,l)} a_i^{(l-1)}] \quad (\because w_{i,j}^{(l-1,l)}, \forall i \in \{1, \dots, n^{(l-1)}\} \text{ 之间相互独立}) \\
\therefore \text{Var}[w_{i,j}^{(l-1,l)} a_i^{(l-1)}] &= \mathbb{E}\left[\left(w_{i,j}^{(l-1,l)} a_i^{(l-1)}\right)^2\right] - \left(\mathbb{E}[w_{i,j}^{(l-1,l)} a_i^{(l-1)}]\right)^2 \quad (\because \text{Var}[\mathcal{X}] = \mathbb{E}[\mathcal{X}^2] - (\mathbb{E}[\mathcal{X}])^2) \\
&= \mathbb{E}\left[\left(w_{i,j}^{(l-1,l)}\right)^2 \left(a_i^{(l-1)}\right)^2\right] - \left(\mathbb{E}[w_{i,j}^{(l-1,l)} a_i^{(l-1)}]\right)^2 \\
&= \mathbb{E}\left[\left(w_{i,j}^{(l-1,l)}\right)^2\right] \mathbb{E}\left[\left(a_i^{(l-1)}\right)^2\right] - \left(\mathbb{E}[w_{i,j}^{(l-1,l)}] \mathbb{E}[a_i^{(l-1)}]\right)^2 \quad (\because w_{i,j}^{(l-1,l)} \text{ 和 } a_i^{(l-1)} \text{ 相互独立}) \\
&= \mathbb{E}\left[\left(w_{i,j}^{(l-1,l)}\right)^2\right] \mathbb{E}\left[\left(a_i^{(l-1)}\right)^2\right] - \left(\mathbb{E}[w_{i,j}^{(l-1,l)}]\right)^2 \left(\mathbb{E}[a_i^{(l-1)}]\right)^2 \quad (\because (ab)^2 = a^2 b^2, a, b \in \mathbb{R}) \\
&= \underbrace{\left(\text{Var}[w_{i,j}^{(l-1,l)}] + \left(\mathbb{E}[w_{i,j}^{(l-1,l)}]\right)^2\right)}_{\mathbb{E}\left[\left(w_{i,j}^{(l-1,l)}\right)^2\right]} \underbrace{\left(\text{Var}[a_i^{(l-1)}] + \left(\mathbb{E}[a_i^{(l-1)}]\right)^2\right)}_{\mathbb{E}\left[\left(a_i^{(l-1)}\right)^2\right]} \\
&\quad - \left(\mathbb{E}[w_{i,j}^{(l-1,l)}]\right)^2 \left(\mathbb{E}[a_i^{(l-1)}]\right)^2 \quad (\because \text{Var}[\mathcal{X}] = \mathbb{E}[\mathcal{X}^2] - (\mathbb{E}[\mathcal{X}])^2 \therefore \mathbb{E}[\mathcal{X}^2] = \text{Var}[\mathcal{X}] + (\mathbb{E}[\mathcal{X}])^2) \\
&= \left(\text{Var}[w_{i,j}^{(l-1,l)}] + 0^2\right) \left(\text{Var}[a_i^{(l-1)}] + 0^2\right) - 0^2 \times 0^2 \quad (\because \mathbb{E}[w_{i,j}^{(l-1,l)}] = 0, \mathbb{E}[a_i^{(l-1)}] = 0) \\
&= \text{Var}[w_{i,j}^{(l-1,l)}] \text{Var}[a_i^{(l-1)}] \\
\therefore \text{Var}[z_j^{(l)}] &= \sum_{i=1}^{n^{(l-1)}} \text{Var}[w_{i,j}^{(l-1,l)} a_i^{(l-1)}] \\
&= \sum_{i=1}^{n^{(l-1)}} \left(\text{Var}[w_{i,j}^{(l-1,l)}] \text{Var}[a_i^{(l-1)}]\right) \\
&= n^{(l-1)} \text{Var}[w_{i,j}^{(l-1,l)}] \text{Var}[a_i^{(l-1)}] \quad (\because \text{所有 } w_{i,j}^{(l-1,l)}, \forall i \in \{1, \dots, n^{(l-1)}\} \text{ 来自相同分布, 同理 } a_i^{(l-1)}) \\
\therefore \text{Var}[a_j^{(l)}] &\approx \text{Var}[z_j^{(l)}] \\
&= n^{(l-1)} \text{Var}[w_{i,j}^{(l-1,l)}] \text{Var}[a_i^{(l-1)}] \quad (\because \text{初始化时 } a_j^{(l)} = f(z_j^{(l)}) \approx z_j^{(l)})
\end{aligned}$$

也就是说, 前向推理过程中, 输入信号 $a_i^{(l-1)}$ 的方差经过该**激活输出**神经元后被**放大或缩小**了 $n^{(l-1)} \text{Var}[w_{i,j}^{(l-1,l)}]$ 倍。为了保障在初始阶段, 信号在经过多层网络传递的稳定性, 即**信号不被过分放大也不被过分减弱**, 需要尽可能地保障经过权重 $w_{i,j}^{l-1,l}$ 加权作用和激活函数 $f(\cdot)$ 激活作用前后的神经元信号 $a_i^{(l-1)}$ 和 $a_j^{(l)}$ 的方差一致, 即:

$$\text{Var}[a_j^{(l)}] = \text{Var}[a_i^{(l-1)}] \implies n^{(l-1)} \text{Var}[w_{i,j}^{(l-1,l)}] = 1 \implies \text{Var}[w_{i,j}^{(l-1,l)}] = \frac{1}{n^{(l-1)}}$$

Analytics of back-propagation 同理反向传播过程中，针对 $\mathcal{L} \rightarrow a_j^{(l)} \rightarrow z_j^{(l)} \rightarrow a_i^{(l-1)} \forall j \in \{1, \dots, n^{(l)}\}$ 链路：

$$\begin{aligned}\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_i^{(l-1)}} &= \sum_{j=1}^{n^{(l)}} \left(\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \cdot \frac{da_j^{(l)}}{dz_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial a_i^{(l-1)}} \right) \quad (\mathcal{D}: \text{mini-batch dataset}) \\ &\approx \sum_{j=1}^{n^{(l)}} \left(\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \cdot 1 \cdot w_{i,j}^{(l-1,l)} \right) \quad (\because \text{初始化时 } a_j^{(l)} = f(z_j^{(l)}) \approx z_j^{(l)}) \\ &= \sum_{j=1}^{n^{(l)}} \left(w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right)\end{aligned}$$

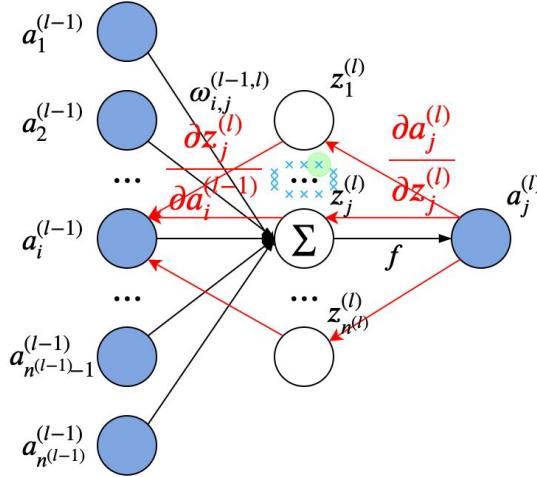


图 111: 反向传播中典型的神经元关系结构

因此在初始化阶段，分析均值：

$$\begin{aligned}\mathbb{E} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_i^{(l-1)}} \right] &= \mathbb{E} \left[\sum_{j=1}^{n^{(l)}} \left(w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right) \right] \\ &= \sum_{j=1}^{n^{(l)}} \mathbb{E} \left[w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \quad (\because \text{反向传播多链路之间相互独立}) \\ &= \sum_{j=1}^{n^{(l)}} \left(\mathbb{E} \left[w_{i,j}^{(l-1,l)} \right] \mathbb{E} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \right) \quad (\because w_{i,j}^{(l-1,l)} \text{ 和 } \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \text{ 相互独立}) \\ &= n^{(l)} \cdot 0 \cdot \mathbb{E} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] = 0 \quad (\because \mathbb{E} \left[w_{i,j}^{(l-1,l)} \right] = 0)\end{aligned}$$

分析方差：

$$\begin{aligned}\text{Var} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_i^{(l-1)}} \right] &= \text{Var} \left[\sum_{j=1}^{n^{(l)}} \left(w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right) \right] \\ &= \sum_{j=1}^{n^{(l)}} \text{Var} \left[w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \quad (\because \text{反向传播多链路相互独立})\end{aligned}$$

针对反向传播中的单一链路：

$$\begin{aligned}
\therefore \text{Var} \left[w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] &= \mathbb{E} \left[\left(\sum_{j=1}^{n^{(l)}} w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right)^2 \right] - \left(\mathbb{E} \left[w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \right)^2 \\
&= \mathbb{E} \left[\left(w_{i,j}^{(l-1,l)} \right)^2 \left(\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right)^2 \right] - \left(\mathbb{E} \left[w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \right)^2 \\
&= \mathbb{E} \left[\left(w_{i,j}^{(l-1,l)} \right)^2 \right] \mathbb{E} \left[\left(\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right)^2 \right] - \left(\mathbb{E} \left[w_{i,j}^{(l-1,l)} \right] \mathbb{E} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \right)^2 \\
&= \underbrace{\left(\text{Var} \left[w_{i,j}^{(l-1,l)} \right] + \left(\mathbb{E} \left[w_{i,j}^{(l-1,l)} \right] \right)^2 \right)}_{\mathbb{E} \left[\left(w_{i,j}^{(l-1,l)} \right)^2 \right]} \underbrace{\left(\text{Var} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] + \left(\mathbb{E} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \right)^2 \right)}_{\mathbb{E} \left[\left(\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right)^2 \right]} \\
&\quad - \left(\mathbb{E} \left[w_{i,j}^{(l-1,l)} \right] \right)^2 \left(\mathbb{E} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \right)^2 \\
&= \left(\text{Var} \left[w_{i,j}^{(l-1,l)} \right] + 0 \right) \left(\text{Var} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] + 0 \right) - 0 \times \left(\mathbb{E} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \right)^2 \\
&= \text{Var} \left[w_{i,j}^{(l-1,l)} \right] \text{Var} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \\
\therefore \text{Var} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_i^{(l-1)}} \right] &= \sum_{j=1}^{n^{(l)}} \text{Var} \left[w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \\
&= \sum_{j=1}^{n^{(l)}} \left(\text{Var} \left[w_{i,j}^{(l-1,l)} \right] \text{Var} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \right) \\
&= n^{(l)} \text{Var} \left[w_{i,j}^{(l-1,l)} \right] \text{Var} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right]
\end{aligned}$$

因此为了保障反向传播信号的稳定性，需要：

$$\begin{aligned}
\text{Var} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_i^{(l-1)}} \right] &= \text{Var} \left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \Rightarrow n^{(l)} \text{Var} \left[w_{i,j}^{(l-1,l)} \right] = 1 \\
&\Rightarrow \text{Var} \left[w_{i,j}^{(l-1,l)} \right] = \frac{1}{n^{(l)}}
\end{aligned}$$

Analytics of both forward-propagation and back-propagation 作为折中，同时考虑信号在正向传播和反向传播中都不被放大或缩小，可以设置：

$$\text{Var} \left[w_{i,j}^{(l-1,l)} \right] = \frac{2}{n^{(l-1)} + n^{(l)}}$$

因此计算出了信号在正向传播和反向传播中都不被放大或缩小时参数理想的方差，而后可以通过高斯分布或均匀分布来随机初始化参数。

- 高斯分布初始化方法：

$$w_{i,j}^{(l-1,l)} \sim \mathcal{N} \left(0, \sqrt{\frac{2}{n^{(l-1)} + n^{(l)}}} \right)$$

- 均匀分布初始化方法：

$$w_{i,j}^{(l-1,l)} \sim \mathcal{U} \left(-\sqrt{\frac{6}{n^{(l-1)} + n^{(l)}}}, \sqrt{\frac{6}{n^{(l-1)} + n^{(l)}}} \right)$$

Proof. 因为随机变量 x 在区间 $[a, b]$ 的方差为：

$$\text{Var}(x) = \frac{(b-a)^2}{12}$$

所以，当采用区间为 $[-r, r]$ 的均匀分布来初始化参数 $w_{i,j}^{(l-1,l)}$ ，并满足 $\text{Var}[w_{i,j}^{(l-1,l)}] = \frac{2}{n^{(l-1)} + n^{(l)}}$ ，那么可解得 r 的取值为：

$$\begin{aligned} \frac{4r^2}{12} &= \frac{2}{n^{(l-1)} + n^{(l)}} \Rightarrow r^2 = \frac{6}{n^{(l-1)} + n^{(l)}} \\ \Rightarrow r &= \sqrt{\frac{6}{n^{(l-1)} + n^{(l)}}} \end{aligned}$$

23.3.2 He initialization

论文出处：[Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification](#)[?]。该方法和的区别是：

- He initialization在 **ReLU** 型激活函数选择下表现的更好。
- Xavier initialization在 **Sigmoid** 型激活函数的选择下表现的更好。

Analytics 考虑 ReLU 型激活函数，形式如下：

$$a_j^{(l)} = f(z_j^{(l)}) = \begin{cases} z_j^{(l)}, & \text{if } z_j^{(l)} > 0 \\ \alpha_j^{(l)} z_j^{(l)}, & \text{if } z_j^{(l)} \leq 0 \end{cases}$$

其中：

- **ReLU**[?]: $\alpha_j^{(l)} = 0$, constant.
- **LeakyReLU**[?]: $\alpha_j^{(l)} > 0$, constant.
- **Parametric-ReLU (PReLU)**[?]: $\alpha_j^{(l)} \in \mathbb{R}$, variable.

Analytics of forward-propagation

Analytics of back-propagation

23.4 Network Normalization

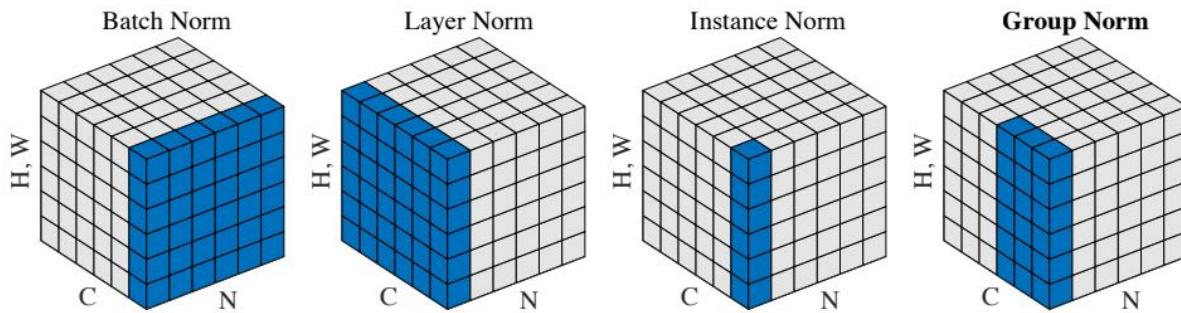


图 112: **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

23.4.1 Batch Normalization (BN)

选自论文 [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#) [?]

Methodology 伪代码如下：

Algorithm 26: Batch Normalizing Transformation[?]

Input: Value of $\mathbf{x} \in \mathbb{R}^d$ over a mini-batch: $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^m$ (d is the number of units, m is the number of batch samples);

Parameters to be learned: $\gamma \in \mathbb{R}^d$, $\beta \in \mathbb{R}^d$.

Output: $\{\mathbf{y}^{(i)} = \text{BN}_{\gamma, \beta}(\mathbf{x}^{(i)})\}_{i=1}^m$, $\mathbf{y}^{(i)} \in \mathbb{R}^d$

1 Calculate mini-batch mean: $\mu_{\mathcal{D}} \leftarrow \frac{1}{m} \sum_{i=1}^m \mathbf{x}^{(i)}$, $\mu_{\mathcal{D}} \in \mathbb{R}^d$

2 Calculate mini-batch variance: $\sigma_{\mathcal{D}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (\mathbf{x}^{(i)} - \mu_{\mathcal{D}})^2$, $\sigma_{\mathcal{D}}^2 \in \mathbb{R}^d$

3 Normalization: $\widehat{\mathbf{x}^{(i)}} \leftarrow \frac{\mathbf{x}^{(i)} - \mu_{\mathcal{D}}}{\sqrt{\sigma_{\mathcal{D}}^2 + \epsilon}}$, $\widehat{\mathbf{x}^{(i)}} \in \mathbb{R}^d$ (Division and square root applied element-wise)

4 Scale & Shift: $\mathbf{y}^{(i)} \leftarrow \gamma \odot \widehat{\mathbf{x}^{(i)}} + \beta \equiv \text{BN}_{\gamma, \beta}(\mathbf{x}^{(i)})$, $\mathbf{y}^{(i)} \in \mathbb{R}^d$

算法解释

- Batch Normalization 是针对列 (batch dataset 中每个独立的 unit) 进行的操作。
- Batch Normalization 的操作：
 - Batch Normalization 先将 batch 中每一列的数据分布转变为标准正态分布，因为如此操作以后，batch 内所有列的数据分布都是正态分布，所以学习率的选择就会更加容易，譬如可以选择更大的学习率。
 - Batch Normalization 然后对每一列加入了再平移参数 β_j 和再缩放参数 γ_j ，使得最终第 $j \in \{1, \dots, d\}$ 个 unit 的分布为 $\mathcal{N} \sim (\beta_j, \gamma_j^2)$ ，这样做是为了保证模型的表达能力不因为规范化而下降。

降。但其实质是防止梯度弥散或梯度爆炸：

$$\because \mathbf{y}^{(i)} = \gamma \odot \widehat{\mathbf{x}^{(i)}} + \beta \quad \therefore \frac{\partial l}{\partial \widehat{\mathbf{x}^{(i)}}} = \frac{\partial l}{\partial \mathbf{y}^{(i)}} \odot \frac{\partial \mathbf{y}^{(i)}}{\partial \widehat{\mathbf{x}^{(i)}}} = \frac{\partial l}{\partial \mathbf{y}^{(i)}} \odot \gamma$$

也就是说：

1. 即使 $\frac{\partial l}{\partial \mathbf{y}^{(i)}}$ 出现了很小的情况（梯度弥散），但也有一定希望通过一个比较大的 γ 调整回来；
2. 即使 $\frac{\partial l}{\partial \mathbf{y}^{(i)}}$ 出现了很大的情况（梯度爆炸），但也有一定希望通过一个比较小的 γ 调整回来。

- Batch Normalization 的理解：

- 每个列的参数（训练的内容为变量） μ_D 和 σ_D^2 是一个 mini-batch 的一阶统计量和二阶统计量；
- 这就要求每一个 mini-batch 的统计量是整体统计量的近似估计，或者说每一个 mini-batch 彼此之间，以及和整体数据，都应该是近似同分布的；
- 分布差距较小的 mini-batches 可以看做是为规范化操作和模型训练引入了噪声，可以增加模型的鲁棒性；
- 但如果每个 mini-batch 的原始分布差别很大，那么不同 mini-batch 的数据将会进行不一样的数据变换，这就增加了模型训练的难度。

- Batch Normalization 的适用场景：

- 每个 mini-batch 中样本数量 m 比较大，且不同 mini-batch 之间的相同 unit 数据分布比较接近。在进行训练之前，要做好充分的 shuffle. 否则效果会差很多。
- 由于 BN 需要在运行过程中统计每个 mini-batch 的一阶统计量和二阶统计量，因此不适用于动态的网络结构和 RNN 网络。

Implement TensorFlow 实现如下：

```
import tensorflow as tf
out = tf.layers.batch_normalization(
    inputs,
    axis=-1,
    momentum=0.99,
    epsilon=0.001,
    center=True,
    scale=True,
    beta_initializer=tf.zeros_initializer(),
    gamma_initializer=tf.ones_initializer(),
    moving_mean_initializer=tf.zeros_initializer(),
    moving_variance_initializer=tf.ones_initializer(),
    beta_regularizer=None,
    gamma_regularizer=None,
    training=False, # training = tf.estimator.ModeKeys.TRAIN
    trainable=True,
    name=None,
)
```

注意易出错忽略的一点是，如果使用该方法定义 Batch Normalization，则需要在训练时写：

```
import tensorflow as tf
loss = ...
if is_training:
    optimizer = tf.train.AdamOptimizer(params.learning_rate)
    global_step = tf.train.get_or_create_global_step()
    if use_batch_norm:
        # Add a dependency to update the moving mean and variance for batch normalization
        with tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_OPS)):
            train_op = optimizer.minimize(loss, global_step=global_step)
    else:
        train_op = optimizer.minimize(loss, global_step=global_step)
```

自定义实现如下：

```
# Copyright (C) 2019 Tong Jia. All rights reserved.
import tensorflow as tf

def batch_norm_2d_fn(x, name_or_scope, is_training, epsilon=1e-6, decay=0.999, dtype_val=tf.float32):
    """Definition of batch normalization for a 2D-tensor.

    Reference:
        Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe et al., 2015]
        (https://arxiv.org/pdf/1502.03167.pdf)

    Parameters
    -----
    :param x: (tensor) -- input tensor with shape [None, D].
    :param name_or_scope: (str) -- name of variable scope.
    :param is_training: (bool) -- indicate whether in train mode.
    :param epsilon: (Optional, float) -- a small float preventing the denominator from being 0
    :param decay: (Optional, float) -- decay rate of exponential moving average of mean and variance.
    :param dtype_val: (Optional, tf.Dtype) -- data type of value variable or constant.

    Returns
    -----
    :return: (tensor) -- [None, D] tensor representing the mini-batch tensor after batch normalization
                      operation.

    """
    with tf.variable_scope(name_or_scope=name_or_scope):
        dim = x.get_shape().as_list()[-1]
        # Define trainable variables
        gamma = tf.get_variable(
            name="gamma",
            shape=[dim],
            dtype=dtype_val,
            initializer=tf.ones_initializer(dtype=dtype_val) # another choice is constant initializer with 0.1
        )
```

```

beta =tf.get_variable(
    name="beta",
    shape=[dim],
    dtype=dtype_val,
    initializer=tf.zeros_initializer(dtype=dtype_val)
)

pop_mean =tf.get_variable(
    name="pop_mean",
    shape=[dim],
    dtype=dtype_val,
    initializer=tf.zeros_initializer(dtype=dtype_val),
    trainable=False
)
pop_var =tf.get_variable(
    name="pop_var",
    shape=[dim],
    dtype=dtype_val,
    initializer=tf.ones_initializer(dtype=dtype_val),
    trainable=False
)
if is_training: # for train mode
    # compute mini-batch mean and variance along the first dim(sample index dim), both batch_mean and
    # batch_var
    # are tensors with shape (D, ).
    batch_mean, batch_var =tf.nn.moments(x=x, axes=[0])
    train_mean_op =tf.assign(ref=pop_mean, value=pop_mean *decay +batch_mean *(1 -decay))
    train_var_op =tf.assign(ref=pop_var, value=pop_var *decay +batch_var *(1 -decay))

    with tf.control_dependencies(control_inputs=[train_mean_op, train_var_op]):
        return tf.nn.batch_normalization(
            x=x,
            mean=batch_mean, variance=batch_var,
            offset=beta, scale=gamma,
            variance_epsilon=epsilon
        )
else: # for eval and infer mode
    return tf.nn.batch_normalization(
        x=x,
        mean=pop_mean, variance=pop_var,
        offset=beta, scale=gamma,
        variance_epsilon=epsilon
    )

```

Reference

- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift[?]
- Understanding Batch Normalization[?]
- Implementing Batch Normalization in Tensorflow

23.4.2 Layer Normalization (LN)

论文出处：[Layer Normalizing\[?\]](#)

Methodology 伪代码如下：

Algorithm 27: Layer Normalizing Transformation[?]

Input: Value of $\mathbf{x} \in \mathbb{R}^d$ over a mini-batch: $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^m$ (d is the number of units, m is the number of batch samples); Parameters to be learned: $\gamma \in \mathbb{R}$, $\beta \in \mathbb{R}$.

Output: $\{\mathbf{y}^{(i)} = \text{BN}_{\gamma, \beta}(\mathbf{x}^{(i)})\}_{i=1}^m$, $\mathbf{y}^{(i)} \in \mathbb{R}^d$

1 Calculate per-sample mean: $\mu^{(i)} = \frac{1}{d} \sum_{j=1}^d x_j^{(i)}$;

$$\boldsymbol{\mu} \in \mathbb{R}^m \leftarrow \left(\frac{1}{d} \sum_{j=1}^d x_j^{(1)}, \dots, \frac{1}{d} \sum_{j=1}^d x_j^{(i)}, \dots, \frac{1}{d} \sum_{j=1}^d x_j^{(m)} \right)$$

2 Calculate per-sample variance: $\sigma^{(i)} = \sqrt{\frac{1}{d} \sum_{j=1}^d (x_j^{(i)} - \mu^{(i)})^2}$;

$$\boldsymbol{\sigma}^2 \in \mathbb{R}^m \leftarrow \left(\frac{1}{d} \sum_{j=1}^d (x_j^{(1)} - \mu^{(1)})^2, \dots, \frac{1}{d} \sum_{j=1}^d (x_j^{(i)} - \mu^{(i)})^2, \dots, \frac{1}{d} \sum_{j=1}^d (x_j^{(m)} - \mu^{(m)})^2 \right)$$

3 Normalization: $\widehat{x}_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu^{(i)}}{\sqrt{(\sigma^{(i)})^2 + \epsilon}}$ (Division and square root applied element-wise)

$$\widehat{\mathbf{x}}^{(i)} \in \mathbb{R}^d \leftarrow \left(\frac{x_1^{(i)} - \mu^{(i)}}{\sqrt{(\sigma^{(i)})^2 + \epsilon}}, \dots, \frac{x_j^{(i)} - \mu^{(i)}}{\sqrt{(\sigma^{(i)})^2 + \epsilon}}, \dots, \frac{x_d^{(i)} - \mu^{(i)}}{\sqrt{(\sigma^{(i)})^2 + \epsilon}} \right)$$

4 Scale & Shift, broadcasting γ & β and element-wise multiplication & addition operations:

$$\mathbf{y}^{(i)} \leftarrow \gamma \odot \widehat{\mathbf{x}}^{(i)} + \beta \equiv \text{LN}_{\gamma, \beta}(\mathbf{x}^{(i)}), \mathbf{y}^{(i)} \in \mathbb{R}^d$$

算法解释

- Layer Normalization 是针对行 (batch dataset 中每个独立 sample 的所有维度) 进行的操作。
- Layer Normalization 的理解：
 - Layer Normalization 对于一整层的神经元训练得到同一个转换——所有的输入都在同一个区间范围内。**如果不同输入特征不属于相似的类别 (比如颜色和大小)，那么 Layer Normalization 的处理可能会降低模型的表达能力。**
 - Layer Normalization 不需要保存 mini-batch 的均值和方差，节省了额外的存储空间。
- Layer Normalization 的适用场景：
 - 可用于小 mini-batch 场景、动态网络场景和 RNN。LN 针对单个训练样本进行，不依赖于其他数据，因此可以避免 Batch Normalization 中受 mini-batch 数据分布影响的问题。
 - 适用于计算机图像领域，特别适用于自然语言处理领域。因为一整层的所有神经元都属于相同类型 (NLP 的语义空间)

23.4.3 Instance Normalization (IN)

论文出处：Instance Normalization: The Missing Ingredient for Fast Stylization[?]，主要针对于图像的风格迁移（style transfer），因为图像各个 channel 的跨空间（height & width）统计信息（e.g. mean, variance）代表了图像的风格（artistic style）。

Idea. 通过分别统计一个 channel 的多个局部区域的统计信息，设计局部自适应的 normalization 方法。

Methodology 假设一个 batch 的图像样本 $x \in \mathbb{R}^{N \times C \times H \times W}$ ，并记其中的一个元素点（像素点）为 $x_{n,c,h,w}$ ，其中：

- h and w span spatial dimensions.
- c is the feature channel (color channel if the input is an RGB image).
- n is the index of the image in the batch.

Algorithm 28: Instance Normalization Transformation for Image Input[?]

Input: Input of a mini-batch dataset: $x \in \mathbb{R}^{N \times C \times H \times W}$;

A small constant ϵ ;

learnable affine parameters: $\gamma \in \mathbb{R}$, $\beta \in \mathbb{R}$.

Output: $y_{n,c,h,w} = \text{IN}_{\gamma,\beta}(x_{n,c,h,w})$, $y_{n,c,h,w} \in \mathbb{R}$.

1 Calculate inside per-sample-per-channel mean $\mu_{n,c}$ ($\forall n \in N, c \in C$):

$$\mu_{n,c} \leftarrow \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W x_{n,c,i,j}$$

2 Calculate inside per-sample-per-channel variance: $\sigma_{n,c}^2$ ($\forall n \in N, c \in C$):

$$\sigma_{n,c}^2 \leftarrow \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W (x_{n,c,i,j} - \mu_{n,c})^2$$

3 Normalization for each element inside mini-batch:

$$\hat{x}_{n,c,h,w} \leftarrow \frac{x_{n,c,h,w} - \mu_{n,c}}{\sqrt{\sigma_{n,c}^2 + \epsilon}}$$

4 Scale & Shift:

$$y_{n,c,h,w} \leftarrow \gamma \cdot \hat{x}_{n,c,h,w} + \beta \equiv \text{IN}_{\gamma,\beta}(x_{n,c,h,w})$$

类似地，Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization[?][?] 论文中为了更好地实现图像的风格迁移而提出了 Adaptive Instance Normalization，即：

$$\text{AdaIN}(x, y) = \sigma_{n,c}(y) \left(\frac{x - \mu_{n,c}(x)}{\sigma_{n,c}(x)} + \mu_{n,c}(y) \right)$$

其中 $x \in \mathbb{R}^{M \times C \times H_{\text{source}} \times W_{\text{source}}}$ 是原始图像输入， $y \in \mathbb{R}^{M \times C \times H_{\text{target}} \times W_{\text{target}}}$ 是风格图像输入。Adaptive Instance Normalization 和 Instance Normalization 唯一的区别是，Adaptive Instance Normalization 中没有训练参数 μ 和 σ ，用于 scale 和 shift 的 $\sigma_{n,c}(y)$ 和 $\mu_{n,c}(y)$ 直接来自于 y 的统计信息。

使用 Instance Normalization 的风格迁移如下：



图 113: Artistic style transfer example of Gatys et al. (2016)[?] method.

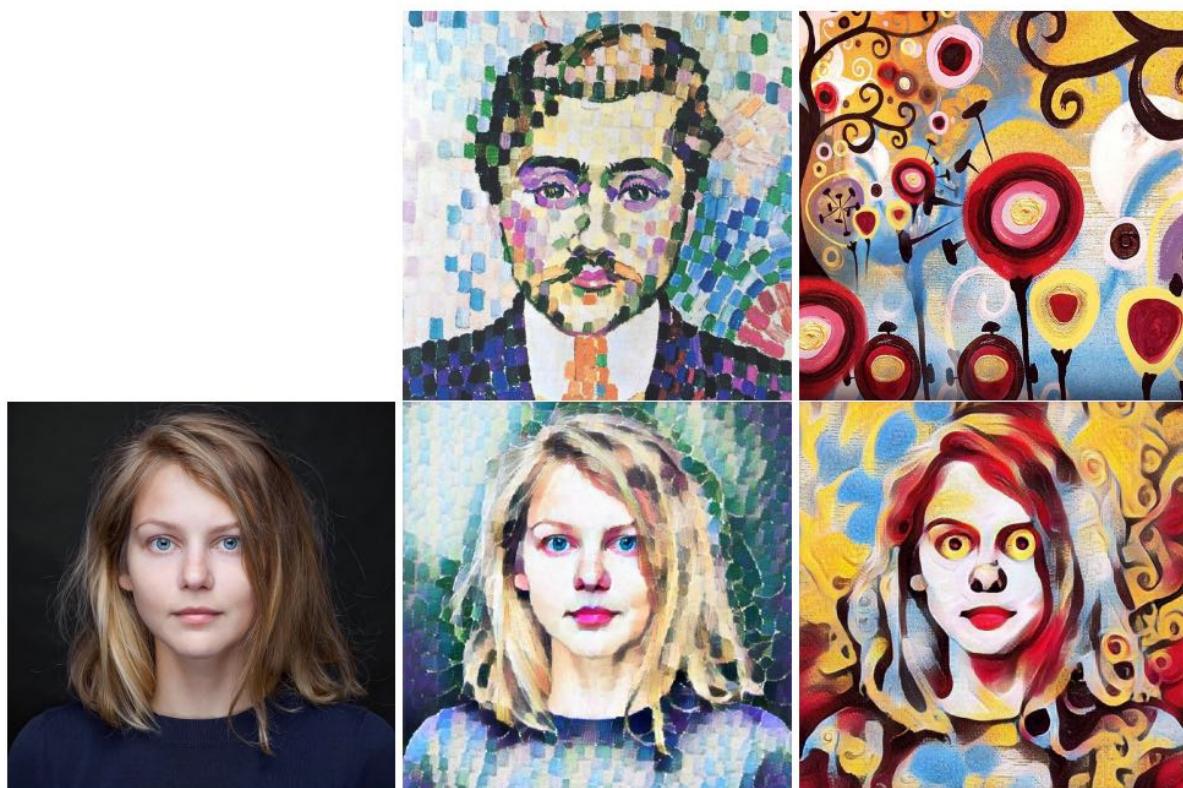


图 114: Stylization examples using proposed method. First row: style images; second row: original image and its stylized versions.

使用 Adaptive Instance Normalization 的风格迁移如下：

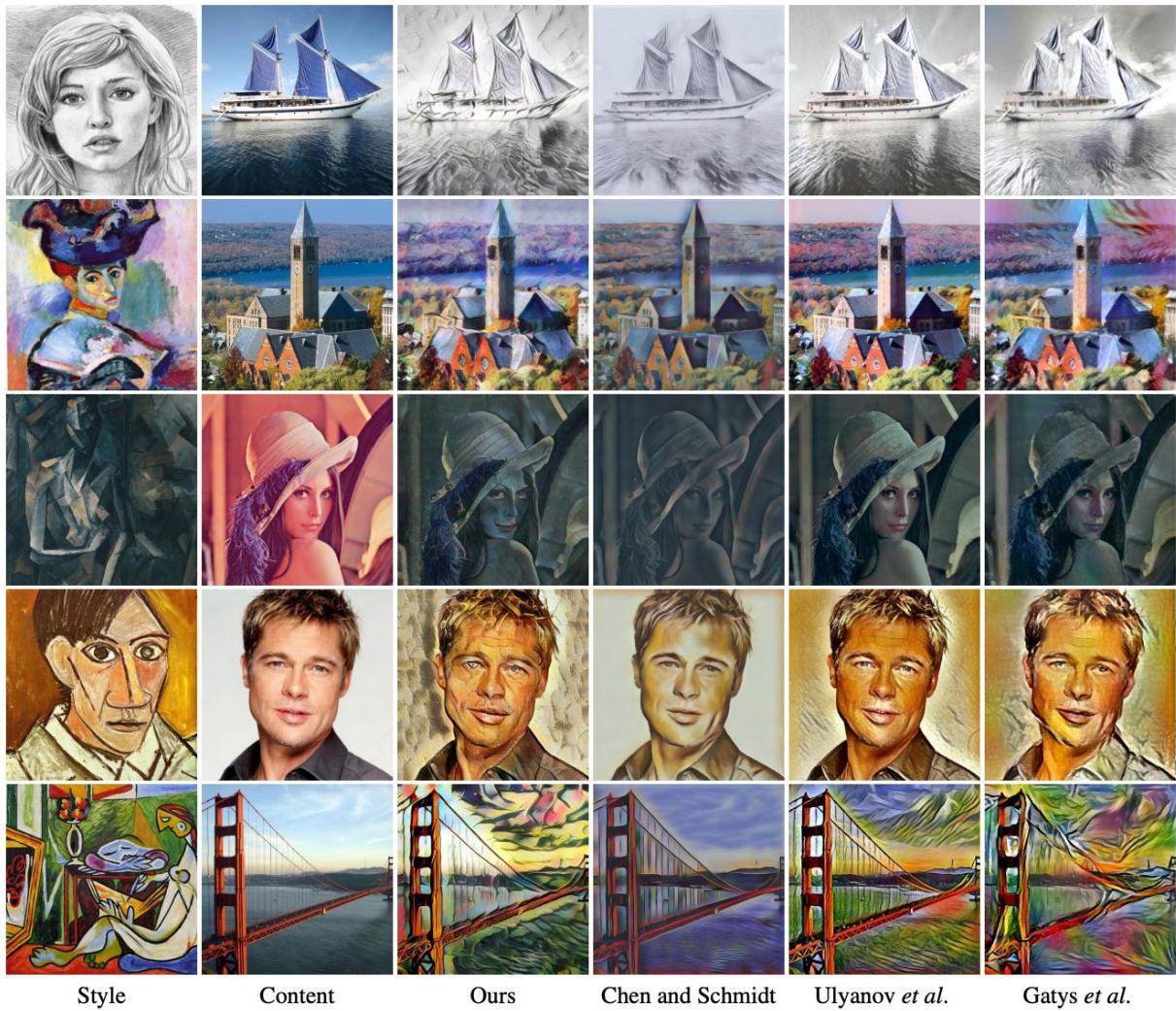
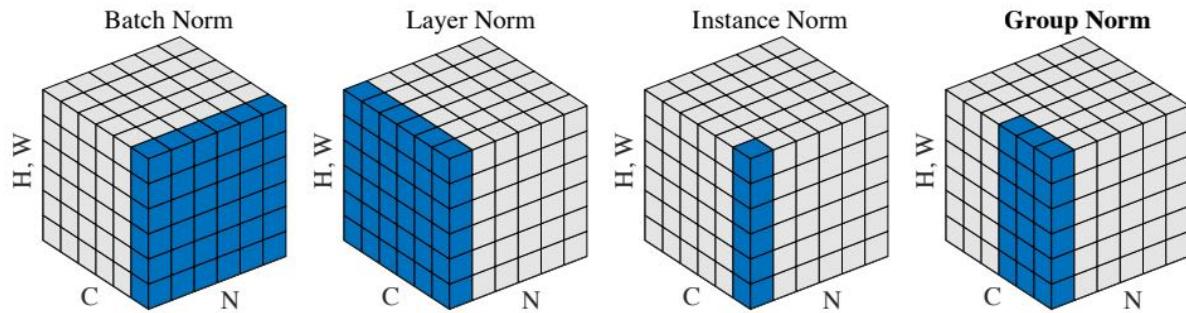


图 115: Example style transfer results. All the tested content and style images are never observed by the network during training.

为了方便对照，针对图像类型数据的 Batch Normalization 和 Layer Normalization 伪代码如下：



Algorithm 29: Batch Normalization Transformation for Image Input[?]**Input:** Input of a mini-batch dataset: $x \in \mathbb{R}^{N \times C \times H \times W}$;A small constant ϵ ;learnable affine parameters: $\gamma \in \mathbb{R}^C$, $\beta \in \mathbb{R}^C$.**Output:** $y_{n,c,h,w} = \text{BN}_{\gamma,\beta}(x_{n,c,h,w})$, $y_{n,c,h,w} \in \mathbb{R}$.**1** Calculate inside per-channel mean μ_c ($\forall c \in C$):

$$\mu_c \leftarrow \frac{1}{N \times H \times W} \sum_{k=1}^N \sum_{i=1}^H \sum_{j=1}^W x_{k,c,i,j}$$

2 Calculate inside per-channel variance: σ_c^2 ($\forall c \in C$):

$$\sigma_c^2 \leftarrow \frac{1}{N \times H \times W} \sum_{k=1}^N \sum_{i=1}^H \sum_{j=1}^W (x_{k,c,i,j} - \mu_c)^2$$

3 Normalization for each element inside mini-batch:

$$\hat{x}_{n,c,h,w} \leftarrow \frac{x_{n,c,h,w} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

4 Scale & Shift:

$$y_{n,c,h,w} \leftarrow \gamma_c \cdot \hat{x}_{n,c,h,w} + \beta_c \equiv \text{BN}_{\gamma,\beta}(x_{n,c,h,w}), \quad \forall c \in C$$

Algorithm 30: Layer Normalization Transformation for Image Input[?]**Input:** Input of a mini-batch dataset: $x \in \mathbb{R}^{N \times C \times H \times W}$;A small constant ϵ ;learnable affine parameters: $\gamma \in \mathbb{R}$, $\beta \in \mathbb{R}$.**Output:** $y_{n,c,h,w} = \text{LN}_{\gamma,\beta}(x_{n,c,h,w})$, $y_{n,c,h,w} \in \mathbb{R}$.**1** Calculate inside per-sample mean μ_n ($\forall n \in N$):

$$\mu_n \leftarrow \frac{1}{C \times H \times W} \sum_{k=1}^C \sum_{i=1}^H \sum_{j=1}^W x_{n,k,i,j}$$

2 Calculate inside per-sample variance: σ_n^2 ($\forall n \in N$):

$$\sigma_n^2 \leftarrow \frac{1}{C \times H \times W} \sum_{k=1}^C \sum_{i=1}^H \sum_{j=1}^W (x_{n,k,i,j} - \mu_n)^2$$

3 Normalization for each element inside mini-batch:

$$\hat{x}_{n,c,h,w} \leftarrow \frac{x_{n,c,h,w} - \mu_n}{\sqrt{\sigma_n^2 + \epsilon}}$$

4 Scale & Shift:

$$y_{n,c,h,w} \leftarrow \gamma \cdot \hat{x}_{n,c,h,w} + \beta \equiv \text{LN}_{\gamma,\beta}(x_{n,c,h,w})$$

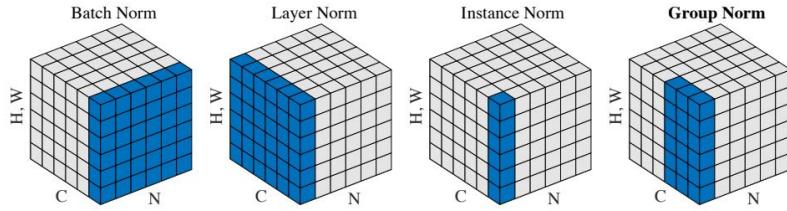
23.4.4 Group Normalization (GN)

论文出处：[Group Normalization\[?\]](#)

23.4.5 Batch-Instance Normalization (BIN)

论文出处：[Batch-Instance Normalization for Adaptively Style-Invariant Neural Networks](#) [?]. 实际上该方法在 Batch Normalization 和 Instance Normalization 之间做了[动态加权平均](#)。

Methodology



Batch Normalization:

$$\begin{aligned}\mu_c^{(B)} &= \frac{1}{N \times H \times W} \sum_{k=1}^N \sum_{i=1}^H \sum_{j=1}^W x_{k,c,i,j} \\ (\sigma_c^{(B)})^2 &= \frac{1}{N \times H \times W} \sum_{k=1}^N \sum_{i=1}^H \sum_{j=1}^W (x_{k,c,i,j} - \mu_c^{(B)})^2 \\ \hat{x}_{n,c,h,w}^{(B)} &= \frac{x_{n,c,h,w} - \mu_c^{(B)}}{\sqrt{(\sigma_c^{(B)})^2 + \epsilon}} \\ y_{n,c,h,w}^{(B)} &= \gamma \cdot \hat{x}_{n,c,h,w}^{(B)} + \beta\end{aligned}$$

Instance Normalization:

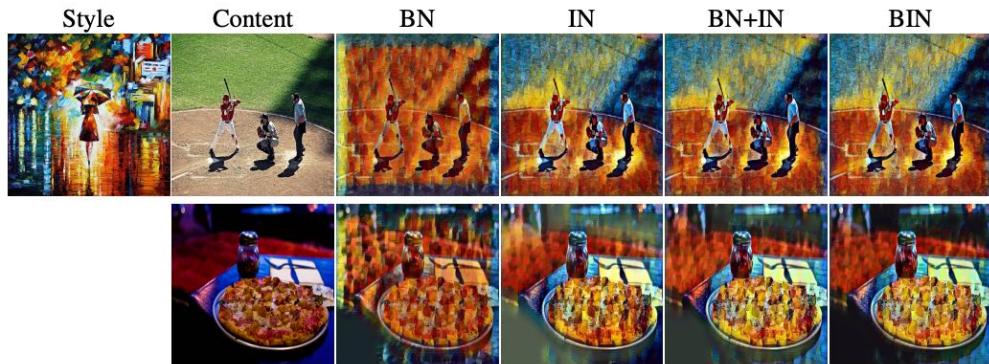
$$\begin{aligned}\mu_{n,c}^{(I)} &= \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W x_{n,c,i,j} \\ (\sigma_{n,c}^{(I)})^2 &= \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W (x_{n,c,i,j} - \mu_{n,c}^{(I)})^2 \\ \hat{x}_{n,c,h,w}^{(I)} &= \frac{x_{n,c,h,w} - \mu_{n,c}^{(I)}}{\sqrt{(\sigma_{n,c}^{(I)})^2 + \epsilon}} \\ y_{n,c,h,w}^{(I)} &= \gamma \cdot \hat{x}_{n,c,h,w}^{(I)} + \beta\end{aligned}$$

Batch Normalization 因为求神经元大小的均值和方差时，考虑了样本 N 的维度，因此：**样本之间的样式差异 (style difference between examples) 得以保留 (通过方差)**；而 Instance Normalization 因为独立地对每个样本的每个 channel 统计均值和方差信息，因此其中不包含样本之间的样式差异信息，只包含单独样本内部一个 channel 内的样式差异信息。而我们希望达到的目的是：对每一个 channel，保留重要的样式属性，而干扰的属性被平滑，因此有了 BIN：

$$y_{n,c,h,w}^{(BI)} = (\rho_c \cdot \hat{x}_{n,c,h,w}^{(B)} + (1 - \rho_c) \cdot \hat{x}_{n,c,h,w}^{(I)}) \cdot \gamma_c + \beta_c$$

其中：

- affine transformation parameters: $\gamma = (\gamma_1, \dots, \gamma_C) \in \mathbb{R}^C$, $\beta = (\beta_1, \dots, \beta_C) \in \mathbb{R}^C$
- learnable parameters: $\rho = (\rho_1, \dots, \rho_C) \in \mathbb{R}^C$. 但是一般会限制 $\rho \in [0, 1]^C$: $\rho \leftarrow \text{clip}_{[0,1]}(\rho - \eta \Delta \rho)$



Algorithm 31: Batch-Instance Normalization Transformation for Image Input[?]

Input: Input of a mini-batch dataset: $x \in \mathbb{R}^{N \times C \times H \times W}$;

A small constant ϵ ;

learnable affine parameters: $\gamma = (\gamma_1, \dots, \gamma_C) \in \mathbb{R}^C$, $\beta = (\beta_c, \dots, \beta_C) \in \mathbb{R}^C$;

learnable parameters: $\rho = (\rho_1, \dots, \rho_C) \in [0, 1]^C$.

Output: $y_{n,c,h,w} = \text{BIN}_{\gamma, \beta, \rho}(x_{n,c,h,w})$, $y_{n,c,h,w} \in \mathbb{R}$.

- 1 Calculate inside per-channel (BN) mean $\mu_c^{(B)}$ ($\forall c \in C$) and inside per-sample-per-channel (IN) mean $\mu_{n,c}^{(I)}$ ($\forall n \in N, c \in C$):

$$\begin{aligned}\mu_c^{(B)} &\leftarrow \frac{1}{N \times H \times W} \sum_{k=1}^N \sum_{i=1}^H \sum_{j=1}^W x_{k,c,i,j} \\ \mu_{n,c}^{(I)} &\leftarrow \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W x_{n,c,i,j}\end{aligned}$$

- 2 Calculate inside per-channel (BN) variance $(\sigma_c^{(B)})^2$ ($\forall c \in C$) and inside per-sample-per-channel (IN) variance $(\sigma_{n,c}^{(I)})^2$ ($\forall n \in N, c \in C$):

$$\begin{aligned}(\sigma_c^{(B)})^2 &\leftarrow \frac{1}{N \times H \times W} \sum_{k=1}^N \sum_{i=1}^H \sum_{j=1}^W (x_{k,c,i,j} - \mu_c^{(B)})^2 \\ (\sigma_{n,c}^{(I)})^2 &\leftarrow \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W (x_{n,c,i,j} - \mu_{n,c}^{(I)})^2\end{aligned}$$

- 3 Normalization for each element inside mini-batch:

$$\hat{x}_{n,c,h,w}^{(B)} \leftarrow \frac{x_{n,c,h,w} - \mu_c^{(B)}}{\sqrt{(\sigma_c^{(B)})^2 + \epsilon}}, \quad \hat{x}_{n,c,h,w}^{(I)} \leftarrow \frac{x_{n,c,h,w} - \mu_{n,c}^{(I)}}{\sqrt{(\sigma_{n,c}^{(I)})^2 + \epsilon}}$$

- 4 Clip the learnable parameter $\rho_c \forall c \in C$:

$$\tilde{\rho}_c \leftarrow \text{clip}_{[0,1]}(\rho_c)$$

- 5 Scale & Shift within each channel $c \in C$:

$$y_{n,c,h,w}^{(BI)} \leftarrow \left(\tilde{\rho}_c \cdot \hat{x}_{n,c,h,w}^{(B)} + (1 - \tilde{\rho}_c) \cdot \hat{x}_{n,c,h,w}^{(I)} \right) \cdot \gamma_c + \beta_c \equiv \text{BIN}_{\gamma, \beta, \rho}(x_{n,c,h,w})$$

Implement BIN 的 TensorFlow 简单实现如下，注意：该实现方法可作为其他 Normalization 方法实现的参考：

```
import tensorflow as tf

def batch_instance_norm(x, scope='batch_instance_norm'):
    # x is a tensor with shape [None, H, W, C]
    with tf.variable_scope(scope):
        ch = x.shape[-1]
        eps = 1e-5

        batch_mean, batch_sigma = tf.nn.moments(x, axes=[0, 1, 2], keep_dims=True)
        x_batch = (x - batch_mean) / (tf.sqrt(batch_sigma + eps))

        ins_mean, ins_sigma = tf.nn.moments(x, axes=[1, 2], keep_dims=True)
        x_ins = (x - ins_mean) / (tf.sqrt(ins_sigma + eps))

        rho = tf.get_variable(name="rho",
                              shape=[ch],
                              dtype=tf.float32,
                              initializer=tf.constant_initializer(1.0),
                              constraint=lambda x: tf.clip_by_value(x, clip_value_min=0.0,
                                                                     clip_value_max=1.0))
        gamma = tf.get_variable(name="gamma",
                               shape=[ch],
                               dtype=tf.float32,
                               initializer=tf.constant_initializer(1.0))
        beta = tf.get_variable("beta", [ch], initializer=tf.constant_initializer(0.0))

        x_hat = rho * x_batch + (1 - rho) * x_ins
        x_hat = x_hat * gamma + beta

    return x_hat
```

23.4.6 Local Response Normalization (LRN)

论文出处：ImageNet Classification with Deep Convolutional Neural Networks (AlexNet)[?]。LRN 是一种收生物学启发的归一化方法，通常用在基于卷积的图像处理问题上。

Methodology 假设一个卷积层的输出特征映射 $\mathbf{Y} \in \mathbb{R}^{H' \times W' \times C'}$ 为三维张量，其中每个切片矩阵（针对 channel 切片） $\mathbf{Y}^{(c)} \in \mathbb{R}^{H' \times W'}$, $\forall c \in C'$ 为一个输出 channel 的特征映射。局部响应归一化 (LRN) 是对临近的 channel 特征映射进行局部归一化，即：

$$\begin{aligned}\hat{Y}_{i,j}^{(c)} &= \frac{Y_{i,j}^{(c)}}{\left(k + \alpha \sum_{l=\max(1, c - \frac{n}{2})}^{\min(C', c + \frac{n}{2})} \left(Y_{i,j}^{(l)} \right)^2 \right)^{\beta}} \\ &\equiv \text{LRN}_{n,k,\alpha,\beta} \left(Y_{i,j}^{(c)} \right)\end{aligned}$$

因此 LRN 可以理解为是对邻近的几个 channel 上相同的切片内位置 (i, j) 进行平滑放大作用。LRN 的所有运算都是 element-wise 的。

Summary

- 局部响应归一化 (LRN) [?] 和层归一化 (LN) [?] 的异同：

– 相同点：

1. 都是对单独样本内的特定神经元 $Y_{i,j}^{(c)}$ 进行归一化。
2. 对 $Y_{i,j}^{(c)}$ 进行归一化时考虑了多个 channels。

– 相异点：

1. **LRN 用于激活函数作用之后** (具体来说：卷积层激活函数之后，汇聚层之前)，LN 用于激活函数作用之前。
2. LN 计算均值，LRN 没有计算均值。
3. LN 考虑了全部 channels，LRN 只考虑邻近部分 channels。

- 局部响应归一化 (LRN) 和生物神经元中的**侧抑制** (lateral inhibition) 现象比较类似，即**活跃神经元对相邻神经元具有抑制作用**。当使用 ReLU 作为激活函数时，神经元的活性值是没有限制的，局部响应归一化可以起到平衡和约束作用。如果一个神经元的活性值 $Y_{i,j}^{(c)}$ 非常大，那么和它邻近的神经元 (相邻近 channels 上相同位置 $Y_{i,j}^{(l)}, l \in \{\max(1, c - \frac{n}{2}), \min(C', c + \frac{n}{2})\}$) 就近似地归一化为 0，从而起到抑制作用，增强模型的泛化能力。

- 局部响应归一化 (LRN) 和最大汇聚 (Max Pooling) 之间的异同：

– 相同点：

1. 都有对邻近神经元的测抑制作用。

– 相异点：

1. LRN 针对几个邻近 channels 上同一个切片内位置 (i, j) 的神经元进行抑制。
2. Max Pooling 针对同一个 channel 上的邻近切片位置的神经元进行抑制。

23.4.7 Gradient Normalization (Gradient Clipping)

Methodology 梯度裁剪 (Gradient Clipping) 是一种有效的防止梯度爆炸 (Gradient Exploding) 的方法

Algorithm 32: Norm clipping the gradients whenever they explode[?]

Compute gradient: $\mathbf{g} \leftarrow \frac{\partial J}{\partial \theta}$

if $\|\mathbf{g}\| \geq threshold$ **then**

$\mathbf{g} \leftarrow \frac{threshold}{\|\mathbf{g}\|} \mathbf{g}$

end

Implement TensorFlow 实现如下：

```
import tensorflow as tf
...
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
grads_and_vars = optimizer.compute_gradients(loss=loss)
for i, (g, v) in enumerate(grads_and_vars):
    if g is not None:
        grads_and_vars[i] = (tf.clip_by_norm(t=g, clip_norm=5), v) # Set threshold = 5
train_op = optimizer.apply_gradients(
    grads_and_vars=grads_and_vars,
    global_step=tf.train.get_global_step()
)
...
...
```

注意：除此之外还有很多 Gradient Clipping 的方法。

23.5 Network Regularization

23.5.1 Label Smoothing Regularization (LSR)

论文出处：[Regularizing Neural Networks by Penalizing Confident Output Distributions\[?\]](#)。该方法用于缓解分类问题中 label 不够 soft 容易导致过拟合的问题，从而提高神经网络的泛化性。

Methodology 将真实概率改造为：

$$p_k = \begin{cases} 1 - \epsilon & \text{if real} \\ \frac{\epsilon}{K-1} & \text{otherwise } (K \text{ is the total number of classes}) \end{cases}$$

譬如，对于一个五分类问题，原始 label 的概率分布为 $t = (0, 0, 1, 0, 0)$ ，而经过 LSR，选择 $\epsilon = 0.01$ ，则平滑后的 label 概率分布为 $y = (0.01, 0.01, 0.96, 0.01, 0.01)$

Note:

- 一个自己的 idea：可将概率改造为概率的概率分布（e.g. Beta 分布）

Reference

- 知乎：[Label Smoothing Regularization \(LSR\) 原理是什么？](#)

23.6 Dropout

23.6.1 Dropout of DNN

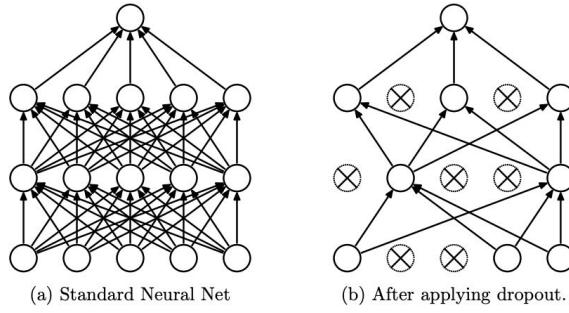


图 116: Neural network with dropout[?]: subsampling of hidden units

Forward propagation of dropout in DNN 多输入-单输出神经元之间的标准网络 (Standard Network) 和 Dropout 网络 (Dropout Network) 之间区别如下 :

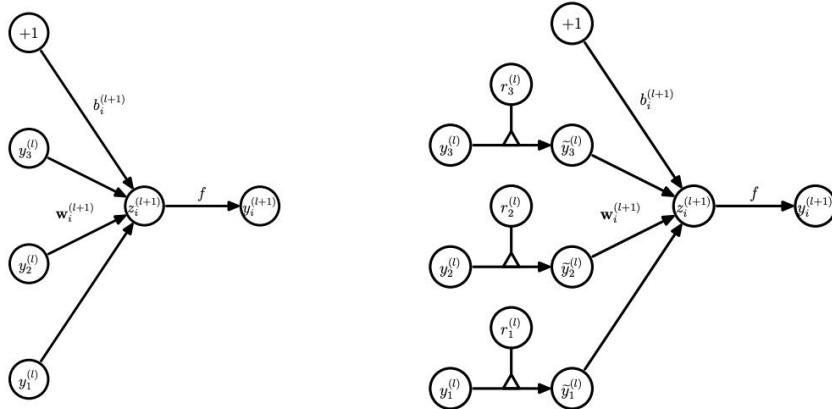


图 117: Comparison of the basic operations of a standard and dropout network. Left figure is standard network and right is dropout network. **Dropout operates before matrix multiplication.**

- 训练阶段 :

标准网络的计算公式 :

$$\begin{aligned} z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \cdot \mathbf{y}^{(l)} + b_i^{(l+1)} \quad (\text{vector-wise}) \\ y_i^{(l+1)} &= f(z_i^{(l+1)}) \end{aligned}$$

采用 Dropout 的网络计算公式 :

$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(p) \\ \tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^l \odot \mathbf{y}^l \quad (\text{element-wise}) \\ z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \cdot \tilde{\mathbf{y}}^{(l)} + b_i^{(l+1)} \\ y_i^{(l+1)} &= f(z_i^{(l+1)}) \end{aligned}$$

- 测试阶段 : 在任意 l 层神经元 y_j^l 与 $l+1$ 层神经元 z_i^{l+1} 的连接中 : 使用 dropout 进行训练时 , y_j^l 以概率 p 处于激活状态 , 并且 $y_j^l \rightarrow y_i^{l+1}$ 的链路上权重为 $w_{i,j}^{l+1}$, 即相当于该权值大小为 $w_{i,j}^{l+1}$ 的链路以概

率 $p \in (0, 1)$ 存在，因此训练阶段该点对点 (unit-to-unit) 链路上的期望权重为：

$$\mathbb{E}_{\text{train}} [w_{i,j}^{l+1}] = p \times 1 \times w_{i,j}^{l+1} + (1-p) \times 0 \times w_{i,j}^{l+1} = p \times w_{i,j}^{l+1}$$

而在测试阶段，因为 y_j^l 以 100% 概率处于激活状态，即该权值大小为 $w_{i,j}^{l+1}$ 的链路以 100% 概率存在，因此测试阶段该点对点 (unit-to-unit) 链路上的期望权重为：

$$\mathbb{E}_{\text{test}} [w_{i,j}^{l+1}] = 1 \times 1 \times w_{i,j}^{l+1} + (1-0) \times 0 \times w_{i,j}^{l+1} = w_{i,j}^{l+1}$$

所以导致：

$$\mathbb{E}_{\text{test}} [w_{i,j}^{l+1}] > \mathbb{E}_{\text{train}} [w_{i,j}^{l+1}] \quad (\text{错误原因 !})$$

因此为了保障在训练阶段和测试阶段该链路的期望权重相同，需要在测试阶段为每个权重做：

$$\mathbf{w}_{\text{test}}^{(l+1)} = p \odot \mathbf{w}^{(l+1)}$$

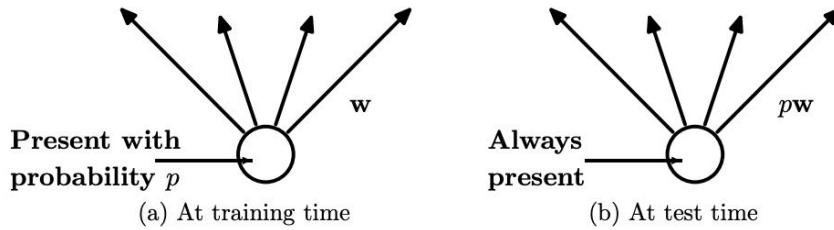


图 118: **Left:** A unit at training phase that is present with probability p and is connected to units in the next layer with weights w . **Right:** At test phase, the unit is always present and the weights are multiplied by p . The output at test time is same as the expected output at training time.

Model implement of dropout in DNN TensorFlow 代码实现如下：

```
import tensorflow as tf
out = tf.nn.dropout(
    x,
    keep_prob, # 1 - keep_prob = dropout_rate
    noise_shape=None,
    seed=None,
    name=None
)
```

Reference

- Dropout: A Simple Way to Prevent Neural Networks from Overfitting[?]

23.6.2 Dropout of CNN

为了捕获局部特征的准确性，建议不要在 CNN (i.e. convolutional layer & pooling layer) 中使用 Dropout。

Reference

- Towards Dropout Training for Convolutional Neural Networks[?]

23.6.3 Dropout of RNN

Forward propagation of dropout in RNN 只丢弃当前时刻输出的参数，不丢弃输入参数，时间轴上的参数。

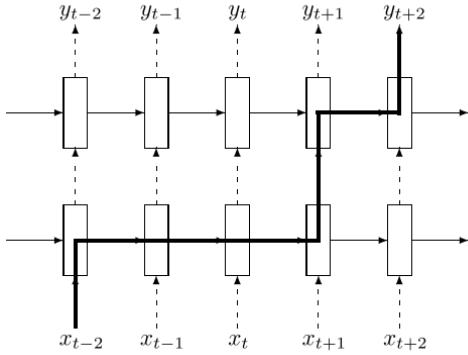


图 119: Dropout in RNN[?]

Model implement of dropout in RNN TensorFlow 代码实现如下：

```

import tensorflow as tf
def rnn_cell(mode):
    # Define such a function to make a basic cell(RNN, LSTM, GRU) with dropout wrapper in model
    # function.
    cell = tf.nn.rnn_cell.LSTMCell(
        num_units=num_hidden_units,
        use_peepholes=False,
        initializer=tf.orthogonal_initializer(dtype=dtype),
        forget_bias=1.0,
        state_is_tuple=True
    )
    if mode == tf.estimator.ModeKeys.TRAIN:
        cell = tf.nn.rnn_cell.DropoutWrapper(
            cell=cell,
            input_keep_prob=1.0,
            output_keep_prob=1 - dropout_rate, # Key point
            state_keep_prob=1.0,
            dtype=dtype
        )
    return cell

```

23.7 Learning Rate Schedules

23.7.1 Learning Rate Decay

学习率衰减 (learning rate decay)，也称为学习率退火 (learning rate annealing) 策略是极其有效的优化函数训练 trick，其主要作用如下：

- 对于非自适应学习率优化算法 (e.g. SGD, Momentum, Nesterov)，其可以调节学习率，使得在训练前期使用较大的学习率保障充分学习、快速收敛；而在训练后期使用较小学习率避免来回震荡，从而无法收敛
- 对于自适应学习率优化算法 (e.g. AdaGrad, RMSProp, Adam)，其（譬如余弦衰减策略）可以增加优化过程的随机性，从而有助于跳出鞍点 (**因此并不是自适应优化算法就没有学习率衰减的需求**)

不失一般性，此处衰减方式设置为按照迭代次数 t 进行衰减，假设初始学习率为 η_0 ，在第 t 次迭代时的学习率为 η_t ，则常见的学习率衰减策略有以下几种：

衰减类型	公式表示
分段常数衰减 (Piecewise Constant Decay)	T_1, \dots, T_m 次迭代后将 η_t 衰减为 η_0 的 β_1, \dots, β_m 倍 ($\beta_m < 1$)
逆时衰减 (Inverse Time Decay)	$\eta_t = \eta_0 \frac{1}{1+\beta \times t}$ ，其中 β 为衰减率
指数衰减 (Exponential Decay)	$\eta_t = \eta_0 \beta^t$ ，其中 $\beta \in (0, 1)$ 为衰减率
自然指数衰减 (Natural Exponential Decay)	$\eta_t = \eta_0 e^{-\beta \times t}$ ，其中 β 为衰减率
余弦衰减 (Cosine Decay)	$\eta_t = \frac{1}{2} \eta_0 \left(1 + \cos\left(\frac{t\pi}{T}\right)\right)$ ，其中 T 为总的迭代次数

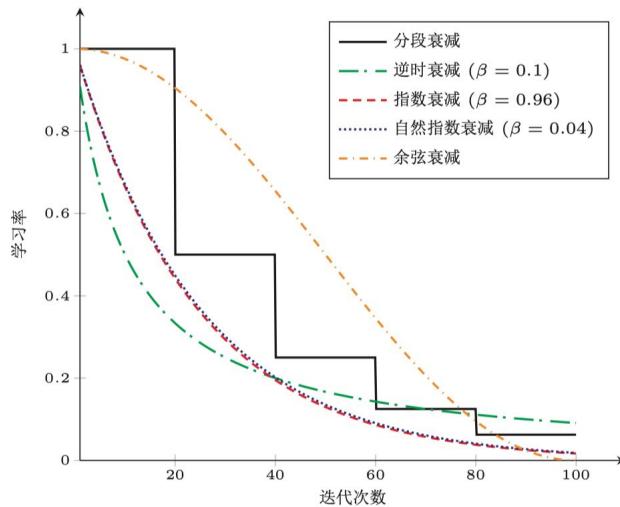


图 120: 不同学习率衰减方法的比较，假设初始学习率 $\eta_0 = 1$ 。

23.7.2 Learning Rate Warmup

初始训练阶段关于学习率大小选择存在的问题：

- 在 mini-batch 梯度下降算法中，在批数据量 (batch size) $|\mathcal{B}|$ 比较大的时候，通常需要较大的学习率 η 。但是由于此时模型参数由随机初始化得到，导致梯度 $\nabla_{\theta}\mathcal{L}(\mathcal{B}; f_{\theta})$ 往往也较大，加之此时学习率 η 也很大，那么会使得**训练非常不稳定**。
- 因此为了在**初始训练阶段**提高训练的稳定性，在最初几轮迭代时，采用较小的学习率，等梯度下降到一定程度（模型参数相对稳定）后再恢复到初始的相对较大学习率。这种方法被称为**学习率预热** (learning rate warmup)。

学习率预热主要分为两大类方法：

1. **Constant Warmup** (瞬间变化): (e.g. ResNet 图像分类 [?] 中先用 0.01 的学习率训练直到训练误差低于 80% (正确率大于 20%)，然后使用 0.1 的学习率进行训练)
2. **Gradual Warmup** (平滑过度): 18 年 Facebook 针对上述 Constant warmup 进行了改进 [?]

- 因为从一个很小的学习率一下子变为一个较大的学习率可能会导致训练误差突然增大
- 所以 gradual warmup 即：从最开始的小学习率开始，每个 iteration 增大一点，直到最初设置的比较大的学习率

预热过程中，每次更新的学习率为：

$$\eta'_t = \frac{t}{T'}\eta_0, \quad 1 \leq t \leq T'$$

其中 T' 为**预热阶段**的总迭代次数， η_0 为**手动设置的较大初始学习率**， η'_t 随着迭代次数 t' 的增加逐步增大；当预热过程结束，再选择一种学习率衰减方法来逐渐降低学习率。

整个训练阶段中学习率大小的变化模式原则应是：

- **前期到中期：越来越大**
- **中起到后期：越来越小**

23.7.3 Periodic Learning Rate Adjustment

Triangular Cyclic Learning Rate

Cosine Decay with Warm Restarts

23.7.4 Linear Scaling Learning Rate

Linear scaling learning rate 是在论文 [?] 中针对比较大的 batch size 而提出的一种方法

- 凸优化问题中，随着批量的增加，收敛速度会降低
- 神经网络也有类似的实证结果。随着 batch size 的增大，处理相同数据量的速度会越来越快，但是达到相同精度所需要的 epoch 数量越来越多
- 使用相同的 epoch 时，大 batch size 训练的模型与小 batch size 训练的模型相比，验证准确率会减小
- 上面提到的 gradual warmup 是解决此问题的方法之一，此外 linear scaling learning rate 也是一种有效的方法
- 在 mini-batch SGD 训练时，梯度下降的值是随机的，因为每一个 batch 的数据是随机选择的。增大 batch size 不会改变梯度的期望，但是会降低它的方差。也就是说，**因为大 batch size 会降低梯度中的噪声，所以我们可以增大学习率来加快收敛**
- 具体做法：e.g. ResNet 原论文 [?] 中 batch size 为 256 时选择的学习率是 0.1，当我们把 batch size 变为一个较大的数 b 时，学习率应该变为 $\eta = 0.1 \times \frac{b}{256}$

23.8 Summary Tricks of Training Neural Networks

23.8.1 Tricks for DNN

23.8.2 Tricks for CNN

根据 Bag of Tricks to Train Convolutional Neural Networks for Image Classification[?] 一文，训练 CNN 做图像分类任务时有如下 Tricks：

23.8.3 Tricks for RNN

Part IV

Reinforcement Learning Model and Theory

24 Reinforcement Learning Basic

24.1 Markov Decision Processes (MDPs)

24.1.1 Infinite-horizon discounted MDPs

In general, Reinforcement Learning(RL) problems can be formulated as Markov Decision Process(MDP) models (**environment models**), and it can be abstracted by a five-tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)[?]$:

- **A finite set of states:** \mathcal{S}
 - Only consider **discrete** and **finite** state spaces here.
- **A finite set of actions:** \mathcal{A}
 - Only consider **discrete** and **finite** action spaces here.
- **Transition function:** $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$
 - $\mathcal{P}(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability of transitioning into state s' upon taking action a in state s , and we always use $\mathcal{P}(s_{t+1}|s_t, a_t)$ to represent transition probability.
- **Reward function:** $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$
 - we always set reward in range of $[0, r_{\max}]$, where $r_{\max} > 0$ is a constant.
 - $r_t = r(s_t, a_t, s_{t+1})$ means an **immediate** reward associated with taking action a_t in state s_t at time-step t , and transitioning into state s_{t+1} .
 - $r(s_t = s, a_t = a, s_{t+1} = s')$ is the immediate reward of transitioning into state s' upon taking action a in state s .
 - $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$, **not** $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, namely r_t belongs to $(s_t \rightarrow a_t \rightarrow s_{t+1})$, not $(s_t \rightarrow a_t)$
- **Discount factor:** $\gamma \in (0, 1)$
 - Lower value of γ place more emphasis on immediate reward.



图 121: [?]The perception-action-learning loop. At time t , the agent receives state s_t from the environment. The agent uses its policy to choose an action a_t . Once the action is executed, the environment transitions a step, providing the next state s_{t+1} as well as feedback in the form of a reward r_t . The agent uses knowledge of state transitions, of the form (s_t, a_t, r_t, s_{t+1}) , in order to learn and improve its policy.

24.1.2 Interaction protocol

In a given MDP $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, the agent interacts with the environment according to the following protocol:

1. the agent starts at some state s_0 ;
2. at each time step $t = 0, 1, \dots$, the agent takes an action $a_t \in \mathcal{A}$, then obtains the immediate reward r_t , and observes the next state s_{t+1} sampled from $s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)$

The interaction record

$$\tau = (s_0^\tau, a_0^\tau, r_0^\tau, s_1^\tau, a_1^\tau, r_1^\tau, s_2^\tau, \dots, s_T^\tau)$$

is called a **trajectory** of length T .

In some situations, it is necessary to specify how the initial state s_0 is generated. We consider s_0 is sampled from an initial distribution $\mu : \mathcal{S} \rightarrow \mathbb{P}$: $s_0 \sim \mu(\cdot)$. When μ is of importance to the discussion, we include it as part of the MDP definition, and write $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \mu)$

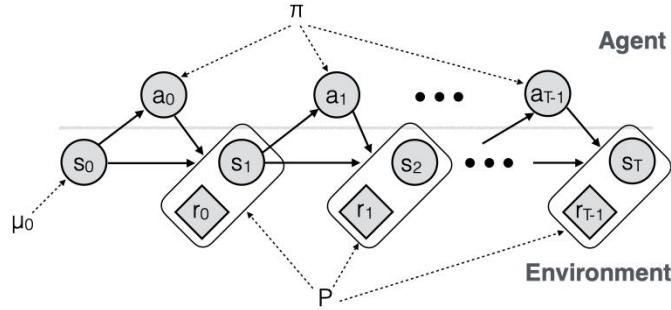


图 122: Illustrate the trajectory of the agent interacts with environment in MDP[?]

24.1.3 Policy and value

Definition 24.1 (deterministic policy). A deterministic policy π is a mapping from state space to action space:

$$\pi : \mathcal{S} \rightarrow \mathcal{A}$$

which the agent chooses an action a_t adaptively based on the current state s_t :

$$a_t = \pi(s_t)$$

Definition 24.2 (stochastic policy). A stochastic policy π is a mapping from state space to a probability distribution over actions:

$$\pi : \mathcal{S} \rightarrow \mathbb{P}^{|\mathcal{A}|}$$

and with a slight abuse of notation we write:

$$a_t \sim \pi(\cdot | s_t)$$

Definition 24.3 (return). The return(rewards-to-go) R_t is the total discounted reward from time-step t .

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

Definition 24.4 (objective of agent). The objective of the agent is to **choose a policy π to maximize the expected discounted sum of rewards, or expected return**:

$$\max_{\pi} \mathbb{E}_{s_0, a_0, s_1, \dots} \left[R_0 = \sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi, \mathcal{P}, \mu \right]$$

The expectation is with respect to the randomness of the observation trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, s_2, \dots)$, that is:

- the randomness of initial state: $s_0 \sim \mu(\cdot)$
- the randomness in state transitions: $s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)$, $\forall t \in \mathbb{N}_0$
- the stochasticity of policy π : $a_t \sim \pi(\cdot | s_t)$, $\forall t \in \mathbb{N}_0$

Lemma 2. Since $r_t \in [0, r_{\max}]$, we have:

$$0 \leq \sum_{t=0}^{\infty} \gamma^t r_t \leq \sum_{t=0}^{\infty} \gamma^t r_{\max} = \frac{r_{\max}}{1 - \gamma}$$

Hence, the discounted sum of future rewards (or the discounted return) along any actual trajectory is always bounded in range $[0, \frac{r_{\max}}{1 - \gamma}]$, and so is its expectation of any form.

24.1.4 Bellman equations

Definition 24.5 (state value function). The state value function $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$ of a MDP is defined as follow:

$$V^\pi(s) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \dots} \left[R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s; \pi, \mathcal{P} \right]$$

which is the expected value of return obtained by following policy π and transition function \mathcal{P} , starting at state s , and:

- $a_{t+k} \sim \pi(\cdot | s_{t+k}), \forall k \in \mathbb{N}_0$
- $s_{t+1+k} \sim \mathcal{P}(\cdot | s_{t+k}, a_{t+k}), \forall k \in \mathbb{N}_0$

Definition 24.6 (state-action value function). The state-action value function(quality function) $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ of a MDP is defined as follow:

$$Q^\pi(s, a) = \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t+2}, \dots} \left[R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a; \pi, \mathcal{P} \right]$$

which is the expected value of return obtained by following policy π and transition function \mathcal{P} , starting at state s and taking action a immediately, and:

- $s_{t+1+k} \sim \mathcal{P}(\cdot | s_{t+k}, a_{t+k}), \forall k \in \mathbb{N}_0$
- $a_{t+k} \sim \pi(\cdot | s_{t+k}), \forall k \in \mathbb{N}_0$

Theorem 3 (interconnection between state value function V^π and state-action value function Q^π). It directly follows from the definitions that state value function and state-action value function are deeply interconnected:

$$\begin{aligned} Q^\pi(s_t, a_t) &= \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} [r_t + \gamma V^\pi(s_{t+1})] \\ V^\pi(s_t) &= \mathbb{E}_{a_t \sim \pi(\cdot | s_t)} Q^\pi(s_t, a_t) \end{aligned}$$



图 123: Illustrate the formula derivation from state-action value function Q^π to state value function V^π

Proof. Firstly we proof the equation: $Q^\pi(\mathbf{s}_t, a_t) = \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} [r(\mathbf{s}_t, a_t, \mathbf{s}_{t+1}) + \gamma V^\pi(\mathbf{s}_{t+1})]$.

$$\begin{aligned}
Q^\pi(\mathbf{s}_t, a_t) &= \mathbb{E}_{\mathbf{s}_{t+1}, a_{t+1}, \mathbf{s}_{t+2}, \dots} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| \mathbf{s}_t, a_t ; \pi, \mathcal{P} \right] \\
&= \mathbb{E}_{\mathbf{s}_{t+1}, a_{t+1}, \mathbf{s}_{t+2}, \dots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \middle| \mathbf{s}_t, a_t ; \pi, \mathcal{P} \right] \\
&= \mathbb{E}_{\mathbf{s}_{t+1}, a_{t+1}, \mathbf{s}_{t+2}, \dots} \left[r_t + \gamma (r_{t+1} + \gamma^1 r_{t+2} + \gamma^2 r_{t+3} + \dots) \middle| \mathbf{s}_t, a_t ; \pi, \mathcal{P} \right] \\
&= \mathbb{E}_{\mathbf{s}_{t+1}, a_{t+1}, \mathbf{s}_{t+2}, \dots} \left[r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \middle| \mathbf{s}_t, a_t ; \pi, \mathcal{P} \right] \text{ (pull out the immediate reward alone)} \\
&= \mathbb{E}_{\mathbf{s}_{t+1}, a_{t+1}, \mathbf{s}_{t+2}, \dots} \left[r_t \middle| \mathbf{s}_t, a_t ; \pi, \mathcal{P} \right] + \mathbb{E}_{\mathbf{s}_{t+1}, a_{t+1}, \mathbf{s}_{t+2}, \dots} \left[\gamma \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \middle| \mathbf{s}_t, a_t ; \pi, \mathcal{P} \right] \\
&= \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} \left[r_t \middle| \mathbf{s}_t, a_t ; \mathcal{P} \right] + \mathbb{E}_{\mathbf{s}_{t+1}, a_{t+1}, \mathbf{s}_{t+2}, \dots} \left[\gamma \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \middle| \mathbf{s}_t, a_t ; \pi, \mathcal{P} \right] \\
&= \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} [r(\mathbf{s}_t, a_t, \mathbf{s}_{t+1})] + \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} \left[\mathbb{E}_{a_{t+1}, \mathbf{s}_{t+2}, a_{t+2}, \dots} \left[\gamma \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \middle| \mathbf{s}_{t+1} ; \pi, \mathcal{P} \right] \right] \\
&= \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} [r(\mathbf{s}_t, a_t, \mathbf{s}_{t+1})] + \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} \underbrace{\left[\gamma \mathbb{E}_{a_{t+1}, \mathbf{s}_{t+2}, a_{t+2}, \dots} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \middle| \mathbf{s}_{t+1} ; \pi, \mathcal{P} \right] \right]}_{\text{definition of } V^\pi(\mathbf{s}_{t+1})} \\
&= \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} [r_t + \gamma V^\pi(\mathbf{s}_{t+1})]
\end{aligned}$$

Similarly, we proof the equation: $V^\pi(\mathbf{s}_t) = \mathbb{E}_{a_t \sim \pi(\cdot | \mathbf{s}_t)} Q^\pi(\mathbf{s}_t, a_t)$ as follow:

$$\begin{aligned}
V^\pi(\mathbf{s}_t) &= \mathbb{E}_{a_t, \mathbf{s}_{t+1}, a_{t+1}, \dots} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| \mathbf{s}_t ; \pi, \mathcal{P} \right] \\
&= \mathbb{E}_{a_t \sim \pi(\cdot | \mathbf{s}_t)} \left[\mathbb{E}_{\mathbf{s}_{t+1}, a_{t+1}, \mathbf{s}_{t+2}, \dots} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| \mathbf{s}_t, a_t ; \pi, \mathcal{P} \right] \right] \\
&\quad \underbrace{\text{definition of } Q^\pi(\mathbf{s}_t, a_t)}_{\text{definition of } Q^\pi(\mathbf{s}_t, a_t)} \\
&= \mathbb{E}_{a_t \sim \pi(\cdot | \mathbf{s}_t)} Q^\pi(\mathbf{s}_t, a_t)
\end{aligned}$$

Theorem 4 (Bellman equation for the state value function). It directly follows the definitions that the Bellman equation of state value function can be formulated as follow:

$$V^\pi(\mathbf{s}_t) = \sum_{a_t \in \mathcal{A}} \pi(a_t | \mathbf{s}_t) \sum_{\mathbf{s}_{t+1} \in \mathcal{S}} \mathcal{P}(\mathbf{s}_{t+1} | \mathbf{s}_t, a_t) (r(\mathbf{s}_t, a_t, \mathbf{s}_{t+1}) + \gamma V^\pi(\mathbf{s}_{t+1}))$$

Proof. Based on the interconnection derivation from state value function to state-action value function: $V^\pi(\mathbf{s}_t) = \mathbb{E}_{a_t \sim \pi(\cdot | \mathbf{s}_t)} Q^\pi(\mathbf{s}_t, a_t)$ and interconnection derivation from state-action value function to state value function $Q^\pi(\mathbf{s}_t, a_t) = \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} [r_t + \gamma V^\pi(\mathbf{s}_{t+1})]$, we can make a derivation as follow:

$$V^\pi(\mathbf{s}_t) = \mathbb{E}_{a_t \sim \pi(\cdot | \mathbf{s}_t)} Q^\pi(\mathbf{s}_t, a_t)$$

$$\begin{aligned}
&= \mathbb{E}_{a_t \sim \pi(\cdot | s_t)} \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} [r_t + \gamma V^\pi(s_{t+1})] \\
&= \sum_{a_t \in \mathcal{A}} \pi(a_t | s_t) \left(\sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1} | s_t, a_t) (r_t + \gamma V^\pi(s_{t+1})) \right) \\
&= \sum_{a_t \in \mathcal{A}} \pi(a_t | s_t) \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1} | s_t, a_t) (r(s_t, a_t, s_{t+1}) + \gamma V^\pi(s_{t+1}))
\end{aligned}$$

Theorem 5 (Bellman equation for the state-action value function). It directly follows the definitions that the Bellman equation of state-action value function can be formulated as follow:

$$Q^\pi(s_t, a_t) = \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1} | s_t, a_t) \left(r(s_t, a_t, s_{t+1}) + \gamma \sum_{a_{t+1} \in \mathcal{A}} \pi(a_{t+1} | s_{t+1}) Q^\pi(s_{t+1}, a_{t+1}) \right)$$

Proof. Based on the interconnection derivation from state-action value function to state value function $Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} [r_t + \gamma V^\pi(s_{t+1})]$ and interconnection derivation from state value function to state-action value function: $V^\pi(s_t) = \mathbb{E}_{a_t \sim \pi(\cdot | s_t)} Q^\pi(s_t, a_t)$, we can make a derivation as follow:

$$\begin{aligned}
Q^\pi(s_t, a_t) &= \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} [r_t + \gamma V^\pi(s_{t+1}) | s_t, a_t; \pi, \mathcal{P}] \\
&= \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} [r_t + \gamma \mathbb{E}_{a_{t+1} \sim \pi(\cdot | s_{t+1})} Q^\pi(s_{t+1}, a_{t+1}) | s_t, a_t; \pi, \mathcal{P}] \\
&= \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1} | s_t, a_t) \left(r_t + \gamma \sum_{a_{t+1} \in \mathcal{A}} \pi(a_{t+1} | s_{t+1}) Q^\pi(s_{t+1}, a_{t+1}) \right) \\
&= \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1} | s_t, a_t) \left(r(s_t, a_t, s_{t+1}) + \gamma \sum_{a_{t+1} \in \mathcal{A}} \pi(a_{t+1} | s_{t+1}) Q^\pi(s_{t+1}, a_{t+1}) \right)
\end{aligned}$$

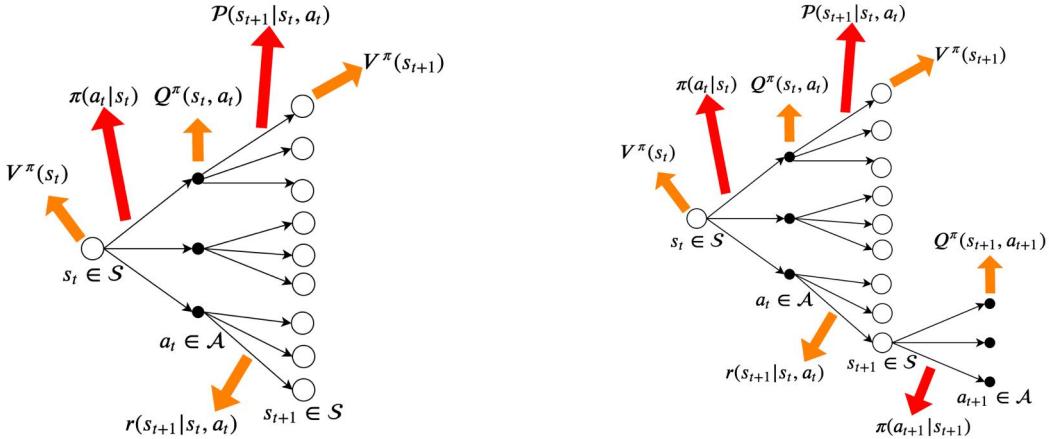


图 125: Illustrate the formula derivation of Bellman equation for state value function V^π

图 126: Illustrate the formula derivation of Bellman equation for state-action value function Q^π

24.1.5 Bellman optimality equations

Definition 24.7 (optimal state value function). The optimal state value function $V^*(\mathbf{s})$ is the maximum state value function over all policies:

$$V^*(\mathbf{s}) = \max_{\pi \in \Pi} V^\pi(\mathbf{s}), \quad \forall \mathbf{s} \in \mathcal{S}$$

Definition 24.8 (optimal state-action value function). The optimal state-action value function (optimal quality function) $Q^*(\mathbf{s}, a)$ is the maximum state-action value function over all policies:

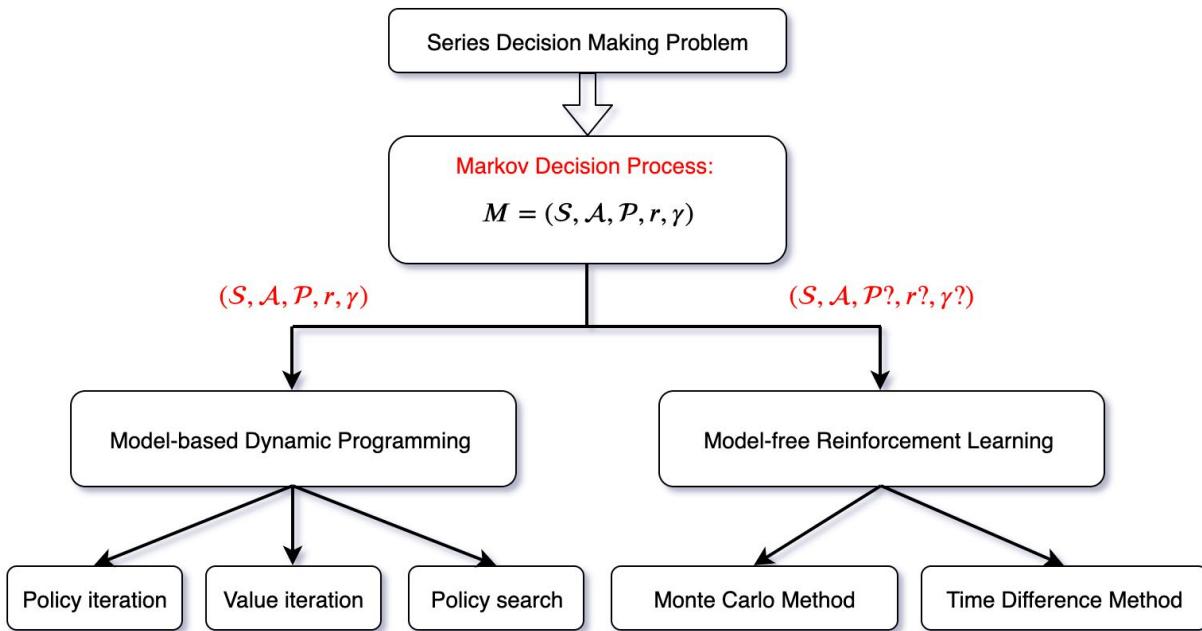
$$Q^*(\mathbf{s}, a) = \max_{\pi \in \Pi} Q^\pi(\mathbf{s}, a), \quad \forall \mathbf{s} \in \mathcal{S}, a \in \mathcal{A}$$

Theorem 6 (interconnection between optimal state value function V^* and optimal state-action value function Q^*). V^* and Q^* satisfy the following set of Bellman optimality equations:

$$\begin{aligned} V^*(\mathbf{s}) &= \max_{a \in \mathcal{A}} Q^*(\mathbf{s}, a) \\ Q^*(\mathbf{s}, a) &= \mathbb{E}_{\mathbf{s}' \sim \mathcal{P}(\cdot | \mathbf{s}, a)} [r(\mathbf{s}, a, \mathbf{s}') + \gamma V^*(\mathbf{s}')] \end{aligned}$$

Theorem 7 (Bellman equation for the optimal state value function). c

24.1.6 Summary



24.1.7 Reference

- CS 598 Statistical Reinforcement Learning (S19)
- Understanding RL: The Bellman Equations
- Lil'Log: A (Long) Peek into Reinforcement Learning

24.2 Semi-Markov Decision Processes (Semi-MDPs) (未完成)

24.3 Partially Observable Markov Decision Processes (POMDPs)

24.3.1 Definition

In general, Partially Observable Markov Decision Processes(POMDPs) can be defined by a siven-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \Omega, \mathcal{O}, \gamma \rangle$, where:

24.3.2 Reference

24.4 Dynamic Programming (DP) in MDPs

从贝尔曼方程可知，如果马尔科夫决策过程 $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$ 的状态转移概率 $\mathcal{P}(s'|s, a)$ 和即时奖励 $r(s, a, s')$ 已知，那么可以直接通过贝尔曼方程来迭代计算其值函数。这种模型已知的强化学习算法也称为**基于模型的强化学习** (Model-based Reinforcement Learning) 算法，这里的**模型**就是指马尔可夫决策过程。

在已知模型时，可以通过动态规划 (Dynamic Programming, DP) 的方法来计算。常用的方法主要有策略迭代 (Policy Iteration) 算法和值迭代 (Value Iteration) 算法。

24.4.1 Value iteration

Algorithm 33: Value Iteration Algorithm

Input: MDPs five-tuple: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$;

A small threshold: $\epsilon > 0$.

Output: Policy $\pi(\cdot) : \mathcal{S} \rightarrow \mathcal{A}$.

1 Initialization: $\forall s \in \mathcal{S}, V(s) = 0$;

2 **repeat**

3 Update value function by Bellman equation:

$$\forall s \in \mathcal{S}, V(s) \leftarrow \max_{a \in \mathcal{A}} \mathbb{E}_{s' \sim \mathcal{P}(\cdot | s, a)} [r(s, a, s') + \gamma V(s')]$$

4 **until** $\forall s \in \mathcal{S}, V(s)$ converges;

5 Compute $Q(s, a)$ based on following equation:

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim \mathcal{P}(\cdot | s, a)} [r(s, a, s') + \gamma V^\pi(s')]$$

6 Get policy $\pi(\cdot)$ based on $Q(s, a)$:

$$\forall s \in \mathcal{S}, \pi(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)$$

24.4.2 Policy iteration

Algorithm 34: Policy Iteration Algorithm

Input: MDPs five-tuple: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$;

A small threshold: $\epsilon > 0$.

Output: Policy $\pi(\cdot) : \mathcal{S} \rightarrow \mathcal{A}$.

1 Initialization: $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}, \pi(a|s) = \frac{1}{|\mathcal{A}|}$;

2 **repeat**

3 // Policy Evaluation

4 **repeat**

5 Compute $V(s)$ by Bellman equation:

$$\forall s \in \mathcal{S}, V(s) \leftarrow \mathbb{E}_{a \sim \pi(\cdot|s)} \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} [r(s, a, s') + \gamma V^\pi(s')]$$

6 **until** $\forall s \in \mathcal{S}, V^\pi(s)$ converges;

7 // Policy Improvement

8 Compute Q -function based on following equation:

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} [r(s, a, s') + \gamma V^\pi(s')]$$

9 Get policy $\pi(\cdot)$ based on $Q(s, a)$:

$$\forall s \in \mathcal{S}, \pi(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)$$

10 **until** $\forall s \in \mathcal{S}, \pi(\cdot|s)$ converges;

24.4.3 Reference

- Value Iteration - Artificial Intelligence: Foundations of Computational Agents

24.5 Applications of Reinforcement Learning Algorithms

24.5.1 Survey

- Reinforcement Learning Applications

24.5.2 Finance

- Deep Reinforcement Learning for Foreign Exchange Trading

24.5.3 Healthcare

- Reinforcement Learning in Healthcare: A Survey

24.5.4 Robotics

24.5.5 Transportation

- Multi-Agent Deep Reinforcement Learning for Large-scale Traffic Signal Control[?]

24.5.6 Recommender Systems

- Deep Reinforcement Learning for List-wise Recommendations[?]

24.6 Frameworks of Reinforcement Learning Algorithms

24.6.1 Commerical

- TF-Agents: A library for Reinforcement Learning in TensorFlow[?]
- OpenAI Baselines: high-quality implementations of reinforcement learning algorithms[?]
- Tensorforce: a TensorFlow library for applied reinforcement learning[?]
- RLlib: Scalable Reinforcement Learning[?]
- Dopamine: a research framework for fast prototyping of reinforcement learning algorithms[?]

24.6.2 Educational

- PyTorch Implementations of Policy Gradient Methods
- Vel (PyTorch)[?]
- RL-Adventure-2 (PyTorch)
- TensorFlow-Model: efficient-hrl
- OpenAI-Spinningup: an educational resource to help anyone learn deep reinforcement learning.
- TRFL: TensorFlow Reinforcement Learning

24.7 Summary of Reinforcement Learning Algorithms

24.7.1 Model-free RL

Algorithm	Policy type	Policy update	State space	Action space
VPG	Policy-based	On-policy	Continuous state space	Discrete action space
DQN	Value-based	Off-policy	Continuous state space	Discrete action space

25 Policy-Based Deep Reinforcement Learning Algorithms

25.1 Vanilla Policy Gradient (VPG)

25.1.1 Introduction

本节选自Policy Gradient Methods for Reinforcement Learning with Function Approximation[?]

Notation

Symbol	Meaning
$s \in \mathcal{S}$	State.
$a \in \mathcal{A}$	Action.
$\mathcal{P}(s' s, a)$	Transition probability of getting to the next state s' from the current state s with action a .
$r(s, a, s')$	Reward.
γ	Discounted factor; penalty to uncertainty of future rewards; $\gamma \in (0, 1)$.
$(s_t, a_t, s_{t+1}, r_t, d)$	Transition tuple at time step t of a trajectory.
$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$	Return, or discounted sum of future rewards.
$\pi(\cdot s)$	Stochastic policy (agent behavior strategy): $\mathcal{S} \rightarrow \mathbb{P}^{ \mathcal{A} }$. $\pi(\cdot s; \theta)$ or $\pi_\theta(\cdot s)$ is a policy parameterized by θ .
$\mu(s)$	Deterministic policy: $\mathcal{S} \rightarrow \mathcal{A}$, we can also label this as $\pi(s)$, but using a different letter gives better distinction so that we can easily tell when the policy is stochastic or deterministic without further explanation. Either π or μ is what a RL algorithm aims to learn.
$V(s)$	State value function measures the expected return under state s
$V^\pi(s)$	The state value under state s when we follow a policy π .
	$V^\pi(s) = \mathbb{E}_{a_t, s_{t+1}, \dots} \left[R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} s_t = s ; \pi, \mathcal{P} \right]$
$Q(s, a)$	State-action value function measures the expected return of a pair of state and action (s, a) .
$Q^\pi(s, a)$	The value of (state, action) pair when we follow a policy π .
	$Q^\pi(s, a) = \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t+2}, \dots} \left[R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} s_t = s, a_t = a ; \pi, \mathcal{P} \right]$
$A^\pi(s, a)$	Advantage function, $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$. it can be considered as another version of Q -value with lower variance by taking the state-value off as the baseline.

25.1.2 Model formulation

Objective The goal of the reinforcement learning is to choose a policy π to maximize the expected return:

$$\begin{aligned} & \max_{\pi_\theta} \mathbb{E}_{s_0, a_0, s_1, \dots} \left[R_0 = \sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi_\theta, \mathcal{P}, \mu \right] \\ \Rightarrow & \max_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_\theta, \mathcal{P}, \mu)} \left[R(\tau) = \sum_{t=0}^{T(\tau)-1} \gamma^t r_t^\tau \right] \end{aligned}$$

The expectation is with respect to the randomness of the observation trajectory $\tau = (s_0^\tau, a_0^\tau, r_0^\tau, s_1^\tau, a_1^\tau, r_1^\tau, s_2^\tau, \dots, s_{T(\tau)}^\tau)$. Assume all trajectories are i.i.d, so we can rewrite the objective as follow:

$$\max_{\theta} \int_{\tau \in \mathcal{T}} R(\tau) \rho(\tau; \pi_\theta, \mathcal{P}, \mu) d\tau$$

Here \mathcal{T} is the trajectory space. And for each trajectory τ , the probability density function of trajectory $\rho(\tau; \pi_\theta, \mathcal{P}, \mu)$ depends on there parameter components:

- $s_0^\tau \sim \mu(\cdot), \forall \tau \in \mathcal{T}$
- $a_t^\tau \sim \pi(\cdot | s_t^\tau; \theta), \forall \tau \in \mathcal{T}, \forall t \in \mathbb{N}_0$
- $s_{t+1}^\tau \sim \mathcal{P}(\cdot | s_t^\tau, a_t^\tau), \forall \tau \in \mathcal{T}, \forall t \in \mathbb{N}_0$

Considering the Markov property on state-action path $(s_0^\tau, a_0^\tau, s_1^\tau, a_1^\tau, s_2^\tau, \dots, s_{T(\tau)}^\tau)$ with length $T(\tau) + 1$, we can rewrite the probability density function of trajectory $\rho(\tau; \pi_\theta, \mathcal{P}, \mu)$ as follow:

$$\begin{aligned} \rho(\tau; \pi_\theta, \mathcal{P}, \mu) &= \mu(s_0^\tau) \cdot \pi(a_0^\tau | s_0^\tau; \theta) \cdot \mathcal{P}(s_1^\tau | s_0^\tau, a_0^\tau) \cdot \pi(a_1^\tau | s_1^\tau; \theta) \cdot \mathcal{P}(s_2^\tau | s_1^\tau, a_1^\tau) \cdot \dots \cdot \mathcal{P}(s_{T(\tau)}^\tau | s_{T(\tau)-1}^\tau, a_{T(\tau)-1}^\tau) \\ &= \mu(s_0^\tau) \cdot \prod_{t=0}^{T(\tau)-1} \pi(a_t^\tau | s_t^\tau; \theta) \cdot \prod_{t=0}^{T(\tau)-1} \mathcal{P}(s_{t+1}^\tau | s_t^\tau, a_t^\tau) \end{aligned}$$

Replace $\rho(\tau; \pi_\theta, \mathcal{P}, \mu)$ in the original objective function with this formula, we can rewrite the objective as follow:

$$\begin{aligned} & \max_{\theta} \int_{\tau \in \mathcal{T}} R(\tau) \left(\mu(s_0^\tau) \cdot \prod_{t=0}^{T(\tau)-1} \pi(a_t^\tau | s_t^\tau; \theta) \cdot \prod_{t=0}^{T(\tau)-1} \mathcal{P}(s_{t+1}^\tau | s_t^\tau, a_t^\tau) \right) d\tau \\ \Rightarrow & \max_{\theta} \int_{\tau \in \mathcal{T}} R(\tau) \log \left(\mu(s_0^\tau) \cdot \prod_{t=0}^{T(\tau)-1} \pi(a_t^\tau | s_t^\tau; \theta) \cdot \prod_{t=0}^{T(\tau)-1} \mathcal{P}(s_{t+1}^\tau | s_t^\tau, a_t^\tau) \right) d\tau \\ \Rightarrow & \max_{\theta} \int_{\tau \in \mathcal{T}} R(\tau) \left(\log \mu(s_0^\tau) + \log \prod_{t=0}^{T(\tau)-1} \pi(a_t^\tau | s_t^\tau; \theta) + \log \prod_{t=0}^{T(\tau)-1} \mathcal{P}(s_{t+1}^\tau | s_t^\tau, a_t^\tau) \right) d\tau \\ \Rightarrow & \max_{\theta} \int_{\tau \in \mathcal{T}} R(\tau) \left(\log \mu(s_0^\tau) + \sum_{t=0}^{T(\tau)-1} \log \pi(a_t^\tau | s_t^\tau; \theta) + \sum_{t=0}^{T(\tau)-1} \log \mathcal{P}(s_{t+1}^\tau | s_t^\tau, a_t^\tau) \right) d\tau \\ \Rightarrow & \max_{\theta} \underbrace{\int_{\tau \in \mathcal{T}} R(\tau) \log \mu(s_0^\tau) d\tau}_{\text{constant}} + \int_{\tau \in \mathcal{T}} R(\tau) \sum_{t=0}^{T(\tau)-1} \log \pi(a_t^\tau | s_t^\tau; \theta) d\tau + \underbrace{\int_{\tau \in \mathcal{T}} R(\tau) \sum_{t=0}^{T(\tau)-1} \log \mathcal{P}(s_{t+1}^\tau | s_t^\tau, a_t^\tau) d\tau}_{\text{constant}} \end{aligned}$$

$$\begin{aligned}
&\Rightarrow \max_{\boldsymbol{\theta}} \int_{\tau \in \mathcal{T}} R(\tau) \sum_{t=0}^{T(\tau)-1} \log \pi(a_t^\tau | \mathbf{s}_t^\tau; \boldsymbol{\theta}) d\tau \\
&\Rightarrow \max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} R(\tau^{(i)}) \sum_{t=0}^{T(\tau^{(i)})-1} \log \pi(a_t^i | \mathbf{s}_t^i; \boldsymbol{\theta}) \quad (\text{sampling } N \text{ trajectories from } \tau \sim \rho(\tau; \pi_{\boldsymbol{\theta}}, \mathcal{P}, \mu)) \\
&\Rightarrow \max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} \left(\sum_{t=0}^{T(\tau^{(i)})-1} \log \pi(a_t^i | \mathbf{s}_t^i; \boldsymbol{\theta}) \right) \left(\sum_{t=0}^{T(\tau^{(i)})-1} r_t^i \right) \\
&\Rightarrow \max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} \left(\sum_{t=0}^{T(\tau^{(i)})-1} \log \pi(a_t^i | \mathbf{s}_t^i; \boldsymbol{\theta}) \right) \underbrace{\left(\sum_{t=0}^{T(\tau^{(i)})-1} r(\mathbf{s}_t^i, a_t^i, \mathbf{s}_{t+1}^i) \right)}_{R(\tau^{(i)})}
\end{aligned}$$

So if we note objective as follow:

$$J(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \log \pi(a_t^i | \mathbf{s}_t^i; \boldsymbol{\theta})$$

we can get the gradient of parameters:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \nabla_{\boldsymbol{\theta}} \log \pi(a_t^i | \mathbf{s}_t^i; \boldsymbol{\theta})$$

we can update parameters in each policy iteration:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

where η is learning rate.

Here we know a stochastic policy π is the function representing a mapping from state to probability distribution over actions:

$$\pi(\cdot | \mathbf{s}; \boldsymbol{\theta}) : \mathcal{S} \rightarrow \mathbb{P}^{|\mathcal{A}|}$$

then we consider that each action a is sampled from its corresponding known action probability distribution \mathbf{p} (e.g. $a = 2$ is sampled from its corresponding action probability distribution $\mathbf{p} = (0.02, 0.01, 0.92, 0.05)$):

$$a \in \mathcal{A} \sim \mathbf{p} \in \mathbb{P}^{|\mathcal{A}|}$$

Then we can convert **maximizing log likelihood** objective to **minimizing KL-divergence** as follow:

$$\begin{aligned}
&\max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \log \pi(a_t^i | \mathbf{s}_t^i; \boldsymbol{\theta}) \\
&\Rightarrow \min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) D_{\text{KL}}(\mathbf{p}_t^i \| \pi(\cdot | \mathbf{s}_t^i; \boldsymbol{\theta})) \\
&\Rightarrow \min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \sum_{k=0}^{|\mathcal{A}|-1} p_{t,k}^i \log \frac{p_{t,k}^i}{\pi_{t,k}^i(\boldsymbol{\theta})} \quad (\pi_{t,k}^i(\boldsymbol{\theta}) = \pi_k(\cdot | \mathbf{s}_t^i; \boldsymbol{\theta}))
\end{aligned}$$

$$\begin{aligned}
&\Rightarrow \min_{\theta} \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \sum_{k=0}^{|\mathcal{A}|-1} p_{t,k}^i (\log p_{t,k}^i - \log \pi_{t,k}^i(\theta)) \\
&\Rightarrow \min_{\theta} \underbrace{\frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \sum_{k=0}^{|\mathcal{A}|-1} p_{t,k}^i \log p_{t,k}^i}_{\text{constant}} - \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \sum_{k=0}^{|\mathcal{A}|-1} p_{t,k}^i \log \pi_{t,k}^i(\theta) \\
&\Rightarrow \min_{\theta} -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \sum_{k=0}^{|\mathcal{A}|-1} p_{t,k}^i \log \pi_{t,k}^i(\theta)
\end{aligned}$$

So we get the final loss function of MC PG:

$$J(\theta) = -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \sum_{k=0}^{|\mathcal{A}|-1} p_{t,k}^i \log \pi_{t,k}^i(\theta)$$

namely a weighted cross-entropy.

problem of policy gradient:

- high variance
- hard to choose learning rate

solution:

- variance reduction: **policy at timestep t' (after) cannot affect reward at time t (before) when $(t < t')$** , therefore we can replace objective as:

$$J(\theta) \approx -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} \log \pi(a_t^i | s_t^i; \theta) \underbrace{\left(\sum_{t'=t}^{T(\tau^{(i)})-1} \gamma^{t'-t} r(s_{t'}^i, a_{t'}^i, s_{t'+1}^i) \right)}_{\text{reward-to-go}}$$

Therefore, **the cumulative discounted reward occur after corresponding state-action pair**

- baselines:

$$\begin{aligned}
&\min_{\theta} -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} \log \pi(a_t^i | s_t^i; \theta) R(\tau^{(i)}) \\
&\Rightarrow \min_{\theta} -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} \log \pi(a_t^i | s_t^i; \theta) (R(\tau^{(i)}) - b)
\end{aligned}$$

where,

$$b = \mathbb{E}_{\tau \sim \rho(\tau; \pi_\theta, \mathcal{P}, \mu)} [R(\tau)] \approx \frac{1}{N} \sum_{i=0}^{N-1} R(\tau^{(i)})$$

1. subtracting a baseline is unbiased in expectation
2. average reward is not the best baseline, but it's pretty good

Training procedure Pseudocode of Policy Gradient with Reward-to-Go is as follow:

Algorithm 35: Policy Gradient with Reward-to-Go

Sample a set of trajectories $\{\tau^{(i)}\}_{i=1}^N$ under policy $\pi(\cdot | \mathbf{s}; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$

Compute gradients wrt objective:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_{\boldsymbol{\theta}} \log \pi(a_t^i | \mathbf{s}_t^i; \boldsymbol{\theta}) \left(\underbrace{\sum_{t'=t}^T \gamma^{t'-t} r(\mathbf{s}_{t'}^i, a_{t'}^i, \mathbf{s}_{t'+1}^i)}_{\text{reward-to-go } \hat{Q}(\mathbf{s}_t^i, a_t^i)} \right) \right)$$

Update parameters of policy:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

25.1.3 Forms of the policy gradient summary

1. **objective of vanilla policy gradient:**

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\boldsymbol{\theta}}, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} \log \pi_{\boldsymbol{\theta}}(a_t^{\tau} | s_t^{\tau}) R(\tau) \right] \\ &\approx \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T(\tau)-1} \log \pi_{\boldsymbol{\theta}}(a_t^{\tau} | s_t^{\tau}) R(\tau) \end{aligned}$$

where $\forall \tau \in \mathcal{D}$ is sampled following distribution $\tau \sim \rho(\tau; \pi_{\boldsymbol{\theta}}, \mathcal{P}, \mu)$.

2. **objective of policy gradient with Monte Carlo baseline:**

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\boldsymbol{\theta}}, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} \log \pi_{\boldsymbol{\theta}}(a_t^{\tau} | s_t^{\tau}) (R(\tau) - b) \right] \\ &\approx \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T(\tau)-1} \log \pi_{\boldsymbol{\theta}}(a_t^{\tau} | s_t^{\tau}) (R(\tau) - b) \end{aligned}$$

where the baseline(global baseline) b can be estimated by Monte Carlo sampling:

$$b = \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\boldsymbol{\theta}}, \mathcal{P}, \mu)} R(\tau) \approx \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} R(\tau)$$

where $\forall \tau \in \mathcal{D}$ is sampled following distribution $\tau \sim \rho(\tau; \pi_{\boldsymbol{\theta}}, \mathcal{P}, \mu)$.

3. **objective of policy gradient with reward-to-go:**

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\boldsymbol{\theta}}, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} \log \pi_{\boldsymbol{\theta}}(a_t^{\tau} | s_t^{\tau}) \left(\underbrace{\sum_{t'=t}^{T(\tau)-1} \gamma^{t'-t} r(s_{t'}^{\tau}, a_{t'}^{\tau}, s_{t'+1}^{\tau})}_{\text{reward-to-go: } \hat{Q}(s_t^{\tau}, a_t^{\tau})} \right) \right] \\ &\approx \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T(\tau)-1} \log \pi_{\boldsymbol{\theta}}(a_t^{\tau} | s_t^{\tau}) \left(\sum_{t'=t}^{T(\tau)-1} \gamma^{t'-t} r(s_{t'}^{\tau}, a_{t'}^{\tau}, s_{t'+1}^{\tau}) \right) \end{aligned}$$

where $\forall \tau \in \mathcal{D}$ is sampled following distribution $\tau \sim \rho(\tau; \pi_{\boldsymbol{\theta}}, \mathcal{P}, \mu)$.

4. **objective of policy gradient with both reward-to-go and Monte Carlo baseline:**

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\boldsymbol{\theta}}, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} \log \pi_{\boldsymbol{\theta}}(a_t^{\tau} | s_t^{\tau}) \left(\sum_{t'=t}^{T(\tau)-1} \gamma^{t'-t} r(s_{t'}^{\tau}, a_{t'}^{\tau}, s_{t'+1}^{\tau}) - b(s_t^{\tau}) \right) \right] \\ &\approx \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T(\tau)-1} \log \pi_{\boldsymbol{\theta}}(a_t^{\tau} | s_t^{\tau}) \left(\sum_{t'=t}^{T(\tau)-1} \gamma^{t'-t} r(s_{t'}^{\tau}, a_{t'}^{\tau}, s_{t'+1}^{\tau}) - b(s_t^{\tau}) \right) \end{aligned}$$

where the baseline(baseline of state) $b(s_t^{\tau})$ can be estimated by Monte Carlo sampling:

$$b(s) = \hat{V}^{\pi}(s) \approx V^{\pi}(s)$$

$$\begin{aligned}
&= \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta}, \mathcal{P}, \mu)} \left[R(\tau) = \sum_{t=0}^{T(\tau)-1} \gamma^t r(s_t^\tau, a_t^\tau, s_{t+1}^\tau) \mid s_0^\tau = s \right] \\
&\approx \frac{1}{|\mathcal{B}|} \sum_{\tau \in \mathcal{B}} [R(\tau) \mid s_0^\tau = s] \\
&\approx \frac{1}{|\mathcal{B}|} \sum_{\tau \in \mathcal{B}} \left[\sum_{t=0}^{T(\tau)-1} \gamma^t r(s_t^\tau, a_t^\tau, s_{t+1}^\tau) \mid s_0^\tau = s \right]
\end{aligned}$$

where $\forall \tau \in \mathcal{B}$ is sampled following distribution $\tau \sim \rho(\tau; \pi_{\theta}, \mathcal{P}, \mu)$. **it requires us to reset the simulator**, namely in order to computing baseline of state s_t^τ , we need to **resample $|\mathcal{B}|$ trajectories starting from specific state s**

5. **objective of policy gradient with on policy action-state value function:**

$$\begin{aligned}
J(\theta) &= \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta}, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} \log \pi_{\theta}(a_t^\tau | s_t^\tau) Q^{\pi_{\theta}}(s_t^\tau, a_t^\tau) \right] \\
&\approx \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T(\tau)-1} \log \pi_{\theta}(a_t^\tau | s_t^\tau) \hat{Q}^{\pi_{\theta}}(s_t^\tau, a_t^\tau; \phi)
\end{aligned}$$

where $Q^{\pi_{\theta}}(s, a)$ is the true underlying value function, and $\hat{Q}^{\pi_{\theta}}(s, a; \phi)$ is the approximate value function with parameters ϕ .

6. **objective of policy gradient with advantage function(indeed value function):**

$$\begin{aligned}
J(\theta) &= \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta}, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} \log \pi_{\theta}(a_t^\tau | s_t^\tau) \left(\sum_{t'=t}^{T(\tau)-1} \gamma^{t'-t} r(s_{t'}^\tau, a_{t'}^\tau, s_{t'+1}^\tau) - b(s_t^\tau) \right) \right] \\
&\approx \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T(\tau)-1} \log \pi_{\theta}(a_t^\tau | s_t^\tau) \left(\sum_{t'=t}^{T(\tau)-1} \gamma^{t'-t} r(s_{t'}^\tau, a_{t'}^\tau, s_{t'+1}^\tau) - b(s_t^\tau) \right) \\
&= \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T(\tau)-1} \log \pi_{\theta}(a_t^\tau | s_t^\tau) (\hat{Q}^{\pi_{\theta}}(s_t^\tau, a_t^\tau) - \hat{V}^{\pi_{\theta}}(s_t^\tau)) \\
&= \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T(\tau)-1} \log \pi_{\theta}(a_t^\tau | s_t^\tau) \hat{A}^{\pi_{\theta}}(s_t^\tau, a_t^\tau) \\
&\approx \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T(\tau)-1} \log \pi_{\theta}(a_t^\tau | s_t^\tau) \underbrace{\left(r(s_t^\tau, a_t^\tau, s_{t+1}^\tau) + \gamma \hat{V}^{\pi_{\theta}}(s_{t+1}^\tau) - \hat{V}^{\pi_{\theta}}(s_t^\tau) \right)}_{\hat{Q}^{\pi_{\theta}}(s_t^\tau, a_t^\tau)}
\end{aligned}$$

So **only need to fit value function $V^{\pi}(s)$!**

Difference between advantage function and reward-to-go minus Monte Carlo baseline:

- advantage function:

$$\hat{A}^{\pi_{\theta}}(s, a) = r(s, a, s') + \gamma \hat{V}^{\pi_{\theta}}(s') - \hat{V}^{\pi_{\theta}}(s)$$

indeed we estimate the value function $V^{\pi_{\theta}}(\cdot)$, **it's biased estimation, but the better this estimate, the lower the variance.**

- reward-to-go minus Monte Carlo baseline:

$$\sum_{t'=t}^{T-1} \gamma^{t'-t} r(\mathbf{s}_{t'}, a_{t'}, \mathbf{s}_{t'+1}) - b(\mathbf{s}_t)$$

it's unbiased, but high variance single-sample estimate.

So until now, the problem is how to fit value function(advantage value function) ?

- (a) Monte Carlo value function

$$\begin{aligned} V^{\pi_\theta}(\mathbf{s}) &\approx \mathbb{E}_{\tau \sim \rho(\tau; \pi_\theta, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} \gamma^t r(\mathbf{s}_t^\tau, a_t^\tau, \mathbf{s}_{t+1}^\tau) \mid \mathbf{s}_0^\tau = \mathbf{s}; \pi_\theta, \mathcal{P} \right] \\ &\approx \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \left[\sum_{t=0}^{T(\tau)-1} \gamma^t r(\mathbf{s}_t^\tau, a_t^\tau, \mathbf{s}_{t+1}^\tau) \mid \mathbf{s}_0^\tau = \mathbf{s} \right] \end{aligned}$$

where $\forall \tau \in \mathcal{D}$ is sampled following distribution $\tau \sim \rho(\tau; \pi_\theta, \mathcal{P}, \mu)$.

- (b) Monte Carlo with value function approximation

- Monte Carlo(MC) target of value function:

$$\begin{aligned} y(\mathbf{s}) &= V^{\pi_\theta}(\mathbf{s}) \\ &= \mathbb{E}_{\tau \sim \rho(\tau; \pi_\theta, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} \gamma^t r(\mathbf{s}_t^\tau, a_t^\tau, \mathbf{s}_{t+1}^\tau) \mid \mathbf{s}_0^\tau = \mathbf{s}; \pi_\theta, \mathcal{P} \right] \\ &\approx \sum_{t=0}^{T-1} \gamma^t r(\mathbf{s}_t, a_t, \mathbf{s}_{t+1}) \Big|_{\mathbf{s}_0 = \mathbf{s}} \quad (\text{directly sample once in current trajectory}) \end{aligned}$$

for time feasibility, sampling only one trajectory starting from state \mathbf{s} , if we can improve sampling efficiency later, we can use expectation of state function belongs to multiple trajectories starting from state \mathbf{s} as Monte Carlo target.

– **problems to be discussed**

- why not use $\frac{1}{N} \sum_{i=1}^N \left[\sum_{t=0}^T \gamma^t r(\mathbf{s}_t^i, a_t^i, \mathbf{s}_{t+1}^i) \Big|_{\mathbf{s}_0^i = \mathbf{s}} \right]$ as Monte Carlo target of \mathbf{s} in value function approximation task, but only use one observation ?
- is there any other method for policy evaluation ?

then we get training dataset with MC target:

$$\mathcal{D} = \left\{ \mathbf{s}, \underbrace{\sum_{t=0}^T \gamma^t r(\mathbf{s}_t, a_t, \mathbf{s}_{t+1})}_{\text{MC target } y} \Big|_{\mathbf{s}_0 = \mathbf{s}, a_t \sim \pi(\cdot | \mathbf{s}_t), \mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} \right\}_{\mathbf{s} \in \tau \sim \rho(\tau; \pi_\theta, \mathcal{P}, \mu)}$$

- Fit value function by regression task on mean squared error loss:

$$\phi = \arg \min_{\phi} l(\phi) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}, y) \in \mathcal{D}} \|\hat{V}^{\pi_\theta}(\mathbf{s}; \phi) - y\|^2$$

(c) Temporal Difference with value function approximation (**used in Actor-Critic**)

- Temporal Difference(TD) target of value function:

$$\begin{aligned}
 y(\mathbf{s}) &= V^{\pi_\theta}(\mathbf{s}) \\
 &= \mathbb{E}_{\tau \sim \rho(\tau; \pi_\theta, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} \gamma^t r(\mathbf{s}_t^\tau, a_t^\tau, \mathbf{s}_{t+1}^\tau) \mid \mathbf{s}_0^\tau = \mathbf{s}; \pi_\theta, \mathcal{P} \right] \\
 &= \mathbb{E}_{\tau \sim \rho(\tau; \pi_\theta, \mathcal{P}, \mu)} \left[r(\mathbf{s}_0^\tau, a_0^\tau, \mathbf{s}_1^\tau) + \sum_{t=1}^{T(\tau)-1} \gamma^t r(\mathbf{s}_t^\tau, a_t^\tau, \mathbf{s}_{t+1}^\tau) \mid \mathbf{s}_0^\tau = \mathbf{s}; \pi_\theta, \mathcal{P} \right] \\
 &= \mathbb{E}_{\tau \sim \rho(\tau; \pi_\theta, \mathcal{P}, \mu)} \left[r(\mathbf{s}_0^\tau, a_0^\tau, \mathbf{s}_1^\tau) + \gamma \sum_{k=0}^{T(\tau)-2} \gamma^k r(\mathbf{s}_{1+k}^\tau, a_{1+k}^\tau, \mathbf{s}_{1+k+1}^\tau) \mid \mathbf{s}_0^\tau = \mathbf{s}; \pi_\theta, \mathcal{P} \right] \\
 &= \mathbb{E}_{a_0 \sim \pi_\theta(\cdot | \mathbf{s}_0)} \mathbb{E}_{\mathbf{s}_1 \sim \mathcal{P}(\cdot | \mathbf{s}_0, a_0)} \left[r(\mathbf{s}_0, a_0, \mathbf{s}_1) + \gamma V^{\pi_\theta}(\mathbf{s}_1) \mid \mathbf{s}_0 = \mathbf{s} \right] \\
 &\approx r(\mathbf{s}, a, \mathbf{s}') + \hat{V}^{\pi_\theta}(\mathbf{s}') \quad (\text{directly sample once in current trajectory}) \\
 &\approx r(\mathbf{s}, a, \mathbf{s}') + \hat{V}^{\pi_\theta}(\mathbf{s}'; \phi) \quad (\text{directly use previous fitted value function})
 \end{aligned}$$

then we get training dataset with TD target:

$$\mathcal{D} = \left\{ \mathbf{s}, \underbrace{r(\mathbf{s}, a, \mathbf{s}') + \hat{V}^{\pi_\theta}(\mathbf{s}'; \phi)}_{\text{TD target } y} \right\}_{\mathbf{s} \in \tau \sim \rho(\tau; \pi_\theta, \mathcal{P}, \mu), \mathbf{s}' \sim \mathcal{P}(\cdot | \mathbf{s}, a)}$$

- Fit value function by regression task on mean squared error loss:

$$\phi = \arg \min_{\phi} l(\phi) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}, y) \in \mathcal{D}} \|\hat{V}^{\pi_\theta}(\mathbf{s}; \phi) - y\|^2$$

25.1.4 Reference

- Lil'Log: Policy Gradient Algorithms
- Part 3: Intro to Policy Optimization - Spinningup.OpenAI
- Paper: High-Dimensional Continuous Control Using Generalized Advantage Estimation[?]
- CS231n-Lecture 14: Reinforcement Learning
- Deep Reinforcement Learning in TensorFlow

25.2 Optimality and Approximation with Policy Gradient Methods in Markov Decision Processes

25.2.1 Introduction

本节选自论文Optimality and Approximation with Policy Gradient Methods in Markov Decision Processes[?]

25.2.2 Reference

- Slides in berkeley: Optimality and Approximation with Policy Gradient Methods in Markov Decision Processes

25.3 Trust Region Policy Optimization (TRPO)

25.3.1 Introduction

本节选自Trust Region Policy Optimization[?]，以及John Schulman的博士论文Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs[?]

Preliminaries Consider an infinite-horizon discounted Markov decision process (MDP) defined by the tuple: $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \mu)$, where:

- \mathcal{S} : A finite set of states.
- \mathcal{A} : A finite set of actions.
- $\mathcal{P}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$, transition probability distribution.
- $r: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$, immediate reward function.
- $\gamma \in (0, 1)$: discount factor.
- $\mu: \mathcal{S} \rightarrow \mathbb{P}$: distribution of the initial state s_0 .

Let π denote a stochastic policy:

$$\pi: \mathcal{S} \rightarrow \mathbb{P}^{|\mathcal{A}|}$$

and let $\eta(\pi)$ denote the expected discounted reward:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, s_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi, \mathcal{P}, \mu \right]$$

where:

- $s_0 \sim \mu(\cdot)$
- $a_t \sim \pi(\cdot | s_t; \theta), \forall t \in \mathbb{N}_0$
- $s_{t+1}^\tau \sim \mathcal{P}(\cdot | s_t^\tau, a_t^\tau), \forall t \in \mathbb{N}_0$

Besides, use the following standard definitions of the state-action value function Q^π , the state value function V^π and the state-action advantage function A^π :

$$\begin{aligned} Q^\pi(s_t, a_t) &= \mathbb{E}_{s_{t+1}, a_{t+1}, \dots} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t, a_t; \pi, \mathcal{P} \right] \\ V^\pi(s_t) &= \mathbb{E}_{a_t, s_{t+1}, \dots} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t; \pi, \mathcal{P} \right] \\ A^\pi(s_t, a_t) &= Q^\pi(s_t, a_t) - V^\pi(s_t) \end{aligned}$$

where:

- $a_t \sim \pi(\cdot | s_t), \forall t \in \mathbb{N}_0$
- $s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t), \forall t \in \mathbb{N}_0$

25.3.2 Model formulation

Step-1: Rewrite $\eta(\pi)$ as an increment We hope that after each update iteration of the policy π , the expected return $\eta(\pi)$ can be monotonically increased. i.e., when the policy iteration is in the following update process:

$$\pi \rightarrow \tilde{\pi}$$

the expected return $\eta(\pi)$ can be monotonically increased:

$$\eta(\tilde{\pi}) \geq \eta(\pi)$$

Slightly rewrite $\eta(\pi)$, it will be written as an incremental as follow:

$$\eta(\tilde{\pi}) = \eta(\pi) + \Delta\eta(\pi, \tilde{\pi})$$

therefore, make a prove that the expected return is monotonically increasing is equivalent to make a proof of increment $\Delta\eta(\pi, \tilde{\pi})$ is nonnegative(positive):

$$\Delta\eta(\pi, \tilde{\pi}) \geq 0$$

Lemma 3. Given two policies $\pi, \tilde{\pi}$, there exists:

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, s_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi}(s_t, a_t) \middle| \tilde{\pi}, \mathcal{P}, \mu \right]$$

This expectation is taken over any trajectories $\tau = (s_0, a_0, s_1, a_1, s_2, \dots)$, and the notation $\mathbb{E}_{s_0, a_0, s_1, \dots} [\cdot | \tilde{\pi}, \mathcal{P}, \mu]$ indicates that actions are sampled from $\tilde{\pi}$ to generate τ .

Proof. First note that:

$$\begin{aligned} A^{\pi}(s_t, a_t) &= Q^{\pi}(s_t, a_t) - V^{\pi}(s_t) \\ &= \mathbb{E}_{s_{t+1}, a_{t+1}, \dots} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t, a_t ; \pi, \mathcal{P} \right] - V^{\pi}(s_t) \\ &= \mathbb{E}_{s_{t+1}, a_{t+1}, \dots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \middle| s_t, a_t ; \pi, \mathcal{P} \right] - V^{\pi}(s_t) \\ &= \mathbb{E}_{s_{t+1}, a_{t+1}, \dots} \left[r_t \middle| s_t, a_t ; \pi, \mathcal{P} \right] + \mathbb{E}_{s_{t+1}, a_{t+1}, \dots} \left[\gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \middle| s_t, a_t ; \pi, \mathcal{P} \right] - V^{\pi}(s_t) \\ &= \mathbb{E}_{s_{t+1}, a_{t+1}, \dots} \left[r_t \middle| s_t, a_t ; \pi, \mathcal{P} \right] + \mathbb{E}_{s_{t+1}, a_{t+1}, \dots} \left[\gamma(r_{t+1} + \gamma^1 r_{t+2} + \gamma^2 r_{t+3} + \dots) \middle| s_t, a_t ; \pi, \mathcal{P} \right] - V^{\pi}(s_t) \\ &= \mathbb{E}_{s_{t+1}, a_{t+1}, \dots} \left[r_t \middle| s_t, a_t ; \pi, \mathcal{P} \right] + \gamma \mathbb{E}_{s_{t+1}, a_{t+1}, \dots} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \middle| s_t, a_t ; \pi, \mathcal{P} \right] - V^{\pi}(s_t) \end{aligned}$$

For the first term, because the immediate reward r_t only depends on local path of a trajectory: (s_t, a_t, s_{t+1}) , which only belongs to the state transition of environment. And also the start point of state transition here is (s_t, a_t) , so here we only consider the transition probability \mathcal{P} , we can rewrite it as follow:

$$\begin{aligned} \mathbb{E}_{s_{t+1}, a_{t+1}, \dots} \left[r_t \middle| s_t, a_t ; \pi, \mathcal{P} \right] &= \mathbb{E}_{s_{t+1}} \left[r_t \middle| s_t, a_t ; \mathcal{P} \right] \\ &= \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} [r(s_t, a_t, s_{t+1})] \end{aligned}$$

For the second term, the sum of discounted return $r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots$ depends on the subsequence of a trajectory: $(\mathbf{s}_{t+1}, a_{t+1}, \mathbf{s}_{t+2}, \dots)$. But the start point here is (\mathbf{s}_t, a_t) , so we must convert the start point from (\mathbf{s}_t, a_t) to \mathbf{s}_{t+1} , then represent the sum of discounted return by a V^π function:

$$\begin{aligned}\mathbb{E}_{\mathbf{s}_{t+1}, a_{t+1}, \dots} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid \mathbf{s}_t, a_t ; \pi, \mathcal{P} \right] &= \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} \left[\mathbb{E}_{a_{t+1}, \mathbf{s}_{t+2}, a_{t+2}, \dots} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid \mathbf{s}_{t+1} ; \pi, \mathcal{P} \right] \right] \\ &= \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} [V^\pi(\mathbf{s}_{t+1})]\end{aligned}$$

So we can rewrite $A^\pi(\mathbf{s}_t, a_t)$ as follow:

$$\begin{aligned}A^\pi(\mathbf{s}_t, a_t) &= \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} [r_t] + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t) \\ &= \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} [r_t] + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} V^\pi(\mathbf{s}_{t+1}) - \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} [V^\pi(\mathbf{s}_t)] \text{ (expect of constant function)} \\ &= \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} [r_t + \gamma V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t)]\end{aligned}$$

So based on the above equation of $A^\pi(\mathbf{s}_t, a_t)$, we can rewrite $\mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(\mathbf{s}_t, a_t) \mid \tilde{\pi}, \mathcal{P}, \mu \right]$ as follow:

$$\mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(\mathbf{s}_t, a_t) \mid \tilde{\pi}, \mathcal{P}, \mu \right] = \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} \left[r_t + \gamma V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t) \mid \tilde{\pi}, \mathcal{P}, \mu \right] \mid \tilde{\pi}, \mathcal{P}, \mu \right]$$

Because $(\mathbf{s}_0, a_0, \mathbf{s}_1, \dots) \sim \langle \mu, \pi, \mathcal{P} \rangle$ contains $\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)$, we can rewrite above equation as follow:

$$\begin{aligned}\mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(\mathbf{s}_t, a_t) \mid \tilde{\pi}, \mathcal{P}, \mu \right] &= \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t (r_t + \gamma V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t)) \mid \tilde{\pi}, \mathcal{P}, \mu \right] \\ &= \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} (\gamma^t r_t + \gamma^{t+1} V^\pi(\mathbf{s}_{t+1}) - \gamma^t V^\pi(\mathbf{s}_t)) \mid \tilde{\pi}, \mathcal{P}, \mu \right] \\ &= \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \tilde{\pi}, \mathcal{P}, \mu \right] + \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^{t+1} V^\pi(\mathbf{s}_{t+1}) \mid \tilde{\pi}, \mathcal{P}, \mu \right] \\ &\quad - \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t V^\pi(\mathbf{s}_t) \mid \tilde{\pi}, \mathcal{P}, \mu \right] \\ &= \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \tilde{\pi}, \mathcal{P}, \mu \right] + \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=1}^{\infty} \gamma^t V^\pi(\mathbf{s}_t) - \sum_{t=0}^{\infty} \gamma^t V^\pi(\mathbf{s}_t) \mid \tilde{\pi}, \mathcal{P}, \mu \right] \\ &= \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \tilde{\pi}, \mathcal{P}, \mu \right] - \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t V^\pi(\mathbf{s}_t) - \sum_{t=1}^{\infty} \gamma^t V^\pi(\mathbf{s}_t) \mid \tilde{\pi}, \mathcal{P}, \mu \right] \\ &= \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \tilde{\pi}, \mathcal{P}, \mu \right] - \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} [V^\pi(\mathbf{s}_0) \mid \tilde{\pi}, \mathcal{P}, \mu]\end{aligned}$$

Consider the first term, based on the definition of expected discounted reward, we can rewrite it as follow:

$$\eta(\tilde{\pi}) = \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \tilde{\pi}, \mathcal{P}, \mu \right]$$

Consider the second term, $V^\pi(\mathbf{s}_0)$ is based on policy π , so it's irrelevant in terms of policy $\tilde{\pi}$, and $V^\pi(\mathbf{s}_0)$ is a constant in terms of $\mathbb{E}_{a_0, \mathbf{s}_1, \dots} [\cdot \mid \tilde{\pi}, \mathcal{P}, \mu]$, according to the definition of state value function, we get:

$$\mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} [V^\pi(\mathbf{s}_0) \mid \tilde{\pi}, \mathcal{P}, \mu] = \mathbb{E}_{\mathbf{s}_0 \sim \mu(\cdot)} [\mathbb{E}_{a_0, \mathbf{s}_1, a_2, \dots} [V^\pi(\mathbf{s}_0) \mid \tilde{\pi}, \mathcal{P}, \mu] \mid \mu]$$

$$\begin{aligned}
&= \mathbb{E}_{s_0 \sim \mu(\cdot)} \left[V^\pi(s_0) \middle| \mu \right] \quad (V^\pi(s_0) \text{ is a constant in terms of } \mathbb{E}_{a_0, s_1, \dots} [\cdot \mid \tilde{\pi}, \mathcal{P}, \mu]) \\
&= \mathbb{E}_{s_0 \sim \mu(\cdot)} \left[\mathbb{E}_{a_0, s_1, a_2, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 ; \pi, \mathcal{P} \right] \middle| \pi, \mathcal{P}, \mu \right] \\
&= \mathbb{E}_{s_0, a_0, s_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \pi, \mathcal{P}, \mu \right] \\
&= \eta(\pi)
\end{aligned}$$

so far we have made a proof of the second term of $\mathbb{E}_{s_0, a_0, s_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \mid \tilde{\pi}, \mathcal{P}, \mu \right]$, so we have made a proof of:

$$\mathbb{E}_{s_0, a_0, s_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \mid \tilde{\pi}, \mathcal{P}, \mu \right] = \eta(\tilde{\pi}) - \eta(\pi)$$

Step-2: Rewrite $\eta(\pi)$ by writing s and a explicitly In the above step, we successfully write $\eta(\pi)$ as incremental format. That is, during the policy iteration, we only need to consider whether the increment $\Delta\eta(\pi, \tilde{\pi})$ is positive or not, and the policy update is for positive increment, otherwise the policy will not be updated.

But the expression of above incremental (i.e., $\mathbb{E}_{s_0, a_0, s_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \mid \tilde{\pi}, \mathcal{P}, \mu \right]$) does not give too much information explicitly, so consider to express the state s and the action a explicitly, we rewrite the above incremental as follow:

$$\begin{aligned}
&\mathbb{E}_{s_0, a_0, s_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \mid \tilde{\pi}, \mathcal{P}, \mu \right] \\
&= \mathbb{E}_{s_0 \sim \mu(\cdot)} \left[\mathbb{E}_{a_0, s_1, a_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \mid \tilde{\pi}, \mathcal{P} \right] \middle| \mu \right] \\
&= \mathbb{E}_{s_0 \sim \mu(\cdot)} \left[\mathbb{E}_{a_0 \sim \tilde{\pi}(\cdot \mid s_0)} \left[\mathbb{E}_{s_1, a_1, s_2, \dots} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \mid \tilde{\pi}, \mathcal{P} \right] \middle| \tilde{\pi} \right] \middle| \mu \right] \\
&= \sum_{s_0 \in \mathcal{S}} \mu(s_0) \sum_{a_0 \in \mathcal{A}} \tilde{\pi}(a_0 \mid s_0) \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \mid \tilde{\pi}, \mathcal{P} \right] \\
&= \sum_{s_0 \in \mathcal{S}} \mu(s_0) \sum_{a_0 \in \mathcal{A}} \tilde{\pi}(a_0 \mid s_0) \mathbb{E}_{s_1, a_1, s_2, \dots} \left[A^\pi(s_0, a_0) + \sum_{t=1}^{\infty} \gamma^t A^\pi(s_t, a_t) \mid \tilde{\pi}, \mathcal{P} \right] \\
&= \sum_{s_0 \in \mathcal{S}} \mu(s_0) \sum_{a_0 \in \mathcal{A}} \tilde{\pi}(a_0 \mid s_0) \mathbb{E}_{s_1, a_1, s_2, \dots} \left[A^\pi(s_0, a_0) \mid \tilde{\pi}, \mathcal{P} \right] + \sum_{s_0 \in \mathcal{S}} \mu(s_0) \sum_{a_0 \in \mathcal{A}} \tilde{\pi}(a_0 \mid s_0) \mathbb{E}_{s_1, a_1, s_2, \dots} \left[\sum_{t=1}^{\infty} \gamma^t A^\pi(s_t, a_t) \mid \tilde{\pi}, \mathcal{P} \right] \\
&= \sum_{s_0 \in \mathcal{S}} \mu(s_0) \sum_{a_0 \in \mathcal{A}} \tilde{\pi}(a_0 \mid s_0) A^\pi(s_0, a_0) + \sum_{s_0 \in \mathcal{S}} \mu(s_0) \sum_{a_0 \in \mathcal{A}} \tilde{\pi}(a_0 \mid s_0) \mathbb{E}_{s_1, a_1, s_2, \dots} \left[\sum_{t=1}^{\infty} \gamma^t A^\pi(s_t, a_t) \mid \tilde{\pi}, \mathcal{P} \right]
\end{aligned}$$

Similarly, we get:

$$\begin{aligned}
&\mathbb{E}_{s_1, a_1, s_2, \dots} \left[\sum_{t=1}^{\infty} \gamma^t A^\pi(s_t, a_t) \mid \tilde{\pi}, \mathcal{P} \right] \\
&= \sum_{s_1 \in \mathcal{S}} \mathcal{P}(s_1 \mid s_0, a_0) \sum_{a_1 \in \mathcal{A}} \tilde{\pi}(a_1 \mid s_1) \gamma A^\pi(s_1, a_1) + \sum_{s_1 \in \mathcal{S}} \mathcal{P}(s_1 \mid s_0, a_0) \sum_{a_1 \in \mathcal{A}} \tilde{\pi}(a_1 \mid s_1) \mathbb{E}_{s_2, a_2, s_3, \dots} \left[\sum_{t=2}^{\infty} \gamma^t A^\pi(s_t, a_t) \mid \tilde{\pi}, \mathcal{P} \right]
\end{aligned}$$

Namely, when $t \geq 1$, we achieve an iterative formula as follow:

$$\begin{aligned} & \mathbb{E}_{\mathbf{s}_t, a_t, \mathbf{s}_t, \dots} \left[\sum_{k=t}^{\infty} \gamma^k A^\pi(\mathbf{s}_k, a_k) \middle| \tilde{\pi}, \mathcal{P} \right] \\ &= \sum_{\mathbf{s}_t \in \mathcal{S}} \mathcal{P}(\mathbf{s}_t | \mathbf{s}_{T(t-1)}, a_{t-1}) \sum_{a_t \in \mathcal{A}} \tilde{\pi}(a_t | \mathbf{s}_t) \gamma^t A^\pi(\mathbf{s}_t, a_t) \\ &+ \sum_{\mathbf{s}_t \in \mathcal{S}} \mathcal{P}(\mathbf{s}_t | \mathbf{s}_{t-1}, a_{t-1}) \sum_{a_t \in \mathcal{A}} \tilde{\pi}(a_t | \mathbf{s}_t) \mathbb{E}_{\mathbf{s}_{t+1}, a_{t+1}, \dots} \left[\sum_{k=t+1}^{\infty} \gamma^k A^\pi(\mathbf{s}_k, a_k) \middle| \tilde{\pi}, \mathcal{P} \right] \end{aligned}$$

and when $t = 0$, we achieve:

$$\begin{aligned} & \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{k=0}^{\infty} \gamma^k A^\pi(\mathbf{s}_k, a_k) \middle| \tilde{\pi}, \mathcal{P}, \mu \right] \\ &= \sum_{\mathbf{s}_0 \in \mathcal{S}} \mu(\mathbf{s}_0) \sum_{a_0 \in \mathcal{A}} \tilde{\pi}(a_0 | \mathbf{s}_0) \gamma A^\pi(\mathbf{s}_0, a_0) + \sum_{\mathbf{s}_0 \in \mathcal{S}} \mu(\mathbf{s}_0) \sum_{a_0 \in \mathcal{A}} \tilde{\pi}(a_0 | \mathbf{s}_0) \mathbb{E}_{\mathbf{s}_1, a_1, \mathbf{s}_2, \dots} \left[\sum_{k=1}^{\infty} \gamma^k A^\pi(\mathbf{s}_k, a_k) \middle| \tilde{\pi}, \mathcal{P} \right] \end{aligned}$$

When $t \rightarrow \infty$, $\gamma^t \rightarrow 0$ so we can get:

$$\mathbb{E}_{\mathbf{s}_{t+1}, a_{t+1}, \dots} \left[\sum_{k=t+1}^{\infty} \gamma^k A^\pi(\mathbf{s}_k, a_k) \middle| \tilde{\pi}, \mathcal{P} \right] \longrightarrow 0$$

So let expand the iteration formula, we can get:

$$\begin{aligned} \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{k=0}^{\infty} \gamma^k A^\pi(\mathbf{s}_k, a_k) \middle| \tilde{\pi}, \mathcal{P}, \mu \right] &= \sum_{t=0}^{\infty} \left(\sum_{\mathbf{s} \in \mathcal{S}} P(\mathbf{s}_t = \mathbf{s} ; \tilde{\pi}, \mathcal{P}, \mu) \sum_{a \in \mathcal{A}} \tilde{\pi}(a | \mathbf{s}) \gamma^t A^\pi(\mathbf{s}, a) \right) \\ &= \sum_{t=0}^{\infty} \sum_{\mathbf{s} \in \mathcal{S}} \gamma^t P(\mathbf{s}_t = \mathbf{s} ; \tilde{\pi}, \mathcal{P}, \mu) \sum_{a \in \mathcal{A}} \tilde{\pi}(a | \mathbf{s}) A^\pi(\mathbf{s}, a) \end{aligned}$$

Let $\rho(\cdot ; \pi, \mathcal{P}, \mu)$ be the (unnormalized) discounted visitation probabilities:

$$\rho(\mathbf{s} ; \pi, \mathcal{P}, \mu) = P(\mathbf{s}_0 = \mathbf{s} ; \pi, \mu) + \gamma P(\mathbf{s}_1 = \mathbf{s} ; \pi, \mathcal{P}) + \gamma^2 P(\mathbf{s}_2 = \mathbf{s} ; \pi, \mathcal{P}) + \dots = \sum_{t=0}^{\infty} \gamma^t P(\mathbf{s}_t = \mathbf{s} ; \pi, \mathcal{P}, \mu)$$

So we can push above formulation as follow:

$$\begin{aligned} \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{k=0}^{\infty} \gamma^k A^\pi(\mathbf{s}_k, a_k) \middle| \tilde{\pi}, \mathcal{P}, \mu \right] &= \sum_{t=0}^{\infty} \sum_{\mathbf{s} \in \mathcal{S}} \gamma^t P(\mathbf{s}_t = \mathbf{s} ; \tilde{\pi}, \mathcal{P}, \mu) \sum_{a \in \mathcal{A}} \tilde{\pi}(a | \mathbf{s}) A^\pi(\mathbf{s}, a) \\ &= \sum_{\mathbf{s} \in \mathcal{S}} \left(\sum_{t=0}^{\infty} \gamma^t P(\mathbf{s}_t = \mathbf{s} ; \tilde{\pi}, \mathcal{P}, \mu) \right) \sum_{a \in \mathcal{A}} \tilde{\pi}(a | \mathbf{s}) A^\pi(\mathbf{s}, a) \\ &= \sum_{\mathbf{s} \in \mathcal{S}} \rho(\mathbf{s} ; \tilde{\pi}, \mathcal{P}, \mu) \sum_{a \in \mathcal{A}} \tilde{\pi}(a | \mathbf{s}) A^\pi(\mathbf{s}, a) \end{aligned}$$

So we get:

$$\begin{aligned} \eta(\tilde{\pi}) - \eta(\pi) &= \mathbb{E}_{\mathbf{s}_0, a_0, \mathbf{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(\mathbf{s}_t, a_t) \middle| \tilde{\pi}, \mathcal{P}, \mu \right] \\ &= \sum_{\mathbf{s} \in \mathcal{S}} \rho(\mathbf{s} ; \tilde{\pi}, \mathcal{P}, \mu) \sum_{a \in \mathcal{A}} \tilde{\pi}(a | \mathbf{s}) A^\pi(\mathbf{s}, a) \end{aligned}$$

$$= \sum_{\mathbf{s} \in \mathcal{S}} \rho(\mathbf{s}; \tilde{\pi}, \mathcal{P}, \mu) \mathbb{E}_{a \sim \tilde{\pi}(\cdot | \mathbf{s})} [A^\pi(\mathbf{s}, a)]$$

This equation implies that for any policy update: $\pi \rightarrow \tilde{\pi}$, a nonnegative expected advantage at every state \mathbf{s} (i.e., $\sum_{a \in \mathcal{A}} \tilde{\pi}(a|\mathbf{s}) A^\pi(\mathbf{s}, a) \geq 0$) is guaranteed to increase the policy η performance.

This implies the classic result that the update performed by exact policy iteration, which uses the deterministic policy: $\tilde{\pi}(\mathbf{s}) = \arg \max_a A^\pi(\mathbf{s}, a)$, improves the policy if there is at least one state-action pair (\mathbf{s}, a) with a positive advantage value (i.e., $A^\pi(\mathbf{s}, a) > 0$), and nonzero state visitation probability (i.e., $\rho(\mathbf{s}; \tilde{\pi}, \mathcal{P}, \mu) \neq 0$), otherwise the algorithm has converged to the optimal policy.

However, in the approximate setting, it will typically be unavoidable that there will be some state \mathbf{s} for which the expected advantage is negative (i.e., $\sum_{a \in \mathcal{A}} \tilde{\pi}(a|\mathbf{s}) A^\pi(\mathbf{s}, a) < 0$) due to estimation and approximation error.

Therefore, based on the sampling states $\mathbf{s} \in \mathcal{D}_S$ under policy $\tilde{\pi}$, we may get the result:

$$\eta(\tilde{\pi}) - \eta(\pi) = \frac{1}{|\mathcal{D}_S|} \sum_{\mathbf{s} \in \mathcal{D}_S} \sum_{a \in \mathcal{A}} \tilde{\pi}(a|\mathbf{s}) A^\pi(\mathbf{s}, a) < 0$$

\mathcal{D}_S is a set of sampled states under policy $\tilde{\pi}$, and $|\mathcal{D}_S|$ is the length of the set \mathcal{D}_S .

Step-3: Remove the dependency of discounted visitation probabilities under the new policy $\tilde{\pi}$

The expression derived in the previous step:

$$\eta(\tilde{\pi}) - \eta(\pi) = \sum_{\mathbf{s} \in \mathcal{S}} \rho(\mathbf{s}; \tilde{\pi}, \mathcal{P}, \mu) \sum_{a \in \mathcal{A}} \tilde{\pi}(a|\mathbf{s}) A^\pi(\mathbf{s}, a)$$

therefore during a policy iteration procedure from π to $\tilde{\pi}$, what we need to do is, **under the new policy $\tilde{\pi}$:**

- for all possible states \mathbf{s} ;
- for all possible actions a that could be taken under the state \mathbf{s} .

compute the value of expected advantage $A^\pi(\mathbf{s}, a)$ **under the old policy π** . If we operate in this way, we should consider the time efficiency of sampling, which contains :

- For $\sum_{a \in \mathcal{A}} \tilde{\pi}(a|\mathbf{s})$: given the state \mathbf{s} , it's time-saving to sample all actions $a \in \mathcal{A}$ under stochastic policy $\tilde{\pi}$ because of the relatively small size of the finite set of actions (i.e., $|\mathcal{A}|$)
- For $\sum_{\mathbf{s} \in \mathcal{S}} \rho(\mathbf{s}; \tilde{\pi}, \mathcal{P}, \mu)$: it's rather a time-consuming process not only to sample all $\mathbf{s} \in \mathcal{S}$ because of the huge size of finite state space $|\mathcal{S}|$, but also to sample a specific state \mathbf{s} in different time-step t , in order to estimate the discounted visitation probability: $\rho(\mathbf{s}; \tilde{\pi}, \mathcal{P}, \mu) = \sum_{t=0}^{\infty} \gamma^t P(\mathbf{s}_t = \mathbf{s}; \tilde{\pi}, \mathcal{P}, \mu)$.

that is, the complex dependency of the the visitation probability $\rho(\mathbf{s}; \tilde{\pi}, \mathcal{P}, \mu)$ under policy $\tilde{\pi}$ makes $\eta(\tilde{\pi}) - \eta(\pi) = \sum_{\mathbf{s} \in \mathcal{S}} \rho(\mathbf{s}; \tilde{\pi}, \mathcal{P}, \mu) \sum_{a \in \mathcal{A}} \tilde{\pi}(a|\mathbf{s}) A^\pi(\mathbf{s}, a)$ difficult to optimize directly. Instead, consider ignoring changes in discount visitation probability of state due to policy updates $\pi \Rightarrow \tilde{\pi}$, introduce a local approximation to η as follow:

$$L^\pi(\tilde{\pi}) = \eta(\pi) + \sum_{\mathbf{s} \in \mathcal{S}} \rho(\mathbf{s}; \pi, \mathcal{P}, \mu) \sum_{a \in \mathcal{A}} \tilde{\pi}(a|\mathbf{s}) A^\pi(\mathbf{s}, a)$$

namely, **$L^\pi(\tilde{\pi})$ uses the visitation probability $\rho(\mathbf{s}; \pi, \mathcal{P}, \mu)$ under old policy π** in incremental, rather than use the visitation probability $\rho(\mathbf{s}; \tilde{\pi}, \mathcal{P}, \mu)$ under new policy $\tilde{\pi}$, just like $\eta(\tilde{\pi})$.

So according to whether the states are sampled under the old policy π or the new policy $\tilde{\pi}$, here we have two versions of incremental in policy iteration from old policy π to new policy $\tilde{\pi}$, i.e., $\Delta(\pi, \tilde{\pi})$:

- the states are sampled under the old policy π :

$$L^\pi(\tilde{\pi}) = \eta(\pi) + \sum_{s \in \mathcal{S}} \rho(s; \pi, \mathcal{P}, \mu) \sum_{a \in \mathcal{A}} \tilde{\pi}(a|s) A^\pi(s, a)$$

- the states are sampled under the new policy $\tilde{\pi}$:

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_{s \in \mathcal{S}} \rho(s; \tilde{\pi}, \mathcal{P}, \mu) \sum_{a \in \mathcal{A}} \tilde{\pi}(a|s) A^\pi(s, a)$$

But what is the difference between $L^\pi(\tilde{\pi})$ and $\eta(\tilde{\pi})$, i.e., $|\eta(\tilde{\pi}) - L^\pi(\tilde{\pi})|$?

Step-4: Bound the difference between $L^\pi(\tilde{\pi})$ and $\eta(\tilde{\pi})$ (未完成) To bound the difference between $L^\pi(\tilde{\pi})$ and $\eta(\tilde{\pi})$, we will bound the difference arising from each timestep t . To do this, we first need to introduce a measure of how much π and $\tilde{\pi}$ agree. Specifically, we'll couple the policies, so that they define a joint distribution over pairs of actions.

Definition 25.1. $(\pi, \tilde{\pi})$ is an α -coupled policy pair if it defines a joint distribution $(a, \tilde{a})|s$, such that $P(a \neq \tilde{a}|s) \leq \alpha$ for all s . π and $\tilde{\pi}$ will denote the marginal distributions of a and \tilde{a} , respectively.

In words, this means that at each state s , $(\pi, \tilde{\pi})$ gives us a pair of actions (a, \tilde{a}) , and these actions differ with probability $\mathbb{P} \leq \alpha$.

Definition 25.2.

25.3.3 Reference

- Jonathan Hui: RL—Trust Region Policy Optimization (TRPO) Explained

25.4 Proximal Policy Optimization (PPO)

25.4.1 Introduction

本节选自 [Proximal Policy Optimization Algorithms](#)[?] 以及 [Emergence of Locomotion Behaviours in Rich Environments](#)[?]

Importance Sampling Importance sampling (IS) refers to a collection of Monte Carlo methods where a mathematical expectation of function $f(\mathbf{x})$ with respect to a target distribution $p(\mathbf{x})$, is approximated by a weighted average of random draws from another distribution $q(\mathbf{x})$.

$$\begin{aligned}\mathbb{E}_{\mathbf{x} \sim p}[f(\mathbf{x})] &= \int_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \\ &= \int_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x}) d\mathbf{x} \ (\forall \mathbf{x} \in \mathcal{X}, q(\mathbf{x}) \neq 0) \\ &= \mathbb{E}_{\mathbf{x} \sim q} \left[f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \right]\end{aligned}$$

Analysis Importance Sampling by mathematical expectation and mathematical variance:

- expectation:
 - theoretical expectation: same

$$\mathbb{E}_{\mathbf{x} \sim p}[f(\mathbf{x})] = \mathbb{E}_{\mathbf{x} \sim q} \left[f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \right]$$

- practical expectation: **different**

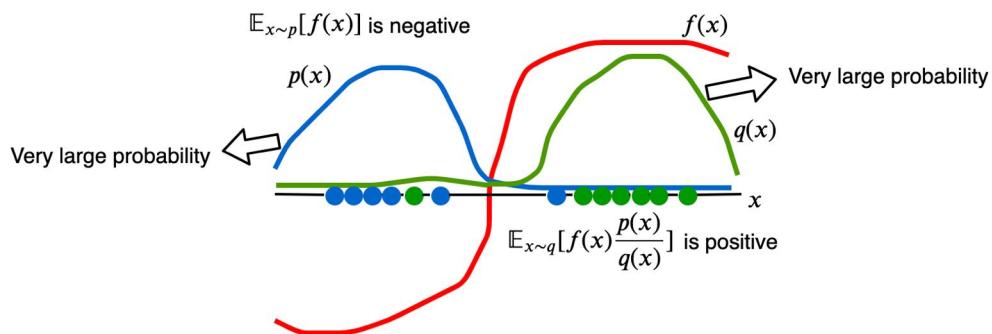


图 127: Importance sampling for a simple practical case, where $f(\mathbf{x})$ is a cost function, both $p(\mathbf{x})$ and $q(\mathbf{x})$ are two different probability density functions in terms of variable \mathbf{x} .

$$\begin{aligned}\mathbb{E}_{\mathbf{x} \sim p}[f(\mathbf{x})] &= \int_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \\ &\approx \frac{1}{N} \sum_{i=0}^{N-1} f(\mathbf{x}^{(i)}) \quad (\text{sampling } N \text{ samples from } \mathbf{x} \sim p) \\ \mathbb{E}_{\mathbf{x} \sim q} \left[f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \right] &= \int_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x}) d\mathbf{x}\end{aligned}$$

$$\approx \frac{1}{N} \sum_{i=0}^{N-1} \left(f(\mathbf{x}^{(i)}) \frac{p(\mathbf{x}^{(i)})}{q(\mathbf{x}^{(i)})} \right) \quad (\text{sampling } N \text{ samples from } \mathbf{x} \sim q)$$

When sampling from the p -probability distribution, the samples are likely to come from the left part of the graph. So it's very likely to happen:

$$\mathbb{E}_{\mathbf{x} \sim p} [f(\mathbf{x})] \approx \frac{1}{N} \sum_{i=0}^{N-1} f(\mathbf{x}^{(i)}) < 0$$

and when sampling from the q -probability distribution, the samples are likely to come from the left part of the graph. So it's very likely to happen:

$$\mathbb{E}_{\mathbf{x} \sim q} \left[f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \right] \approx \frac{1}{N} \sum_{i=0}^{N-1} \left(f(\mathbf{x}^{(i)}) \frac{p(\mathbf{x}^{(i)})}{q(\mathbf{x}^{(i)})} \right) > 0$$

even though $\frac{p(\mathbf{x}^{(i)})}{q(\mathbf{x}^{(i)})}$ is a very small positive number. Therefore it's very likely to happen in practical that:

$$\mathbb{E}_{\mathbf{x} \sim p} [f(\mathbf{x})] \neq \mathbb{E}_{\mathbf{x} \sim q} \left[f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \right]$$

when the two probability distributions p and q are comparatively different.

- variance:

- theoretical variance: **different**

$$\text{Var}_{\mathbf{x} \sim p} [f(\mathbf{x})] = \mathbb{E}_{\mathbf{x} \sim p} \left[(f(\mathbf{x}))^2 \right] - (\mathbb{E}_{\mathbf{x} \sim p} [f(\mathbf{x})])^2 \quad (\text{based on the definition of variance})$$

$$= \int_{\mathbf{x} \in \mathcal{X}} (f(\mathbf{x}))^2 p(\mathbf{x}) d\mathbf{x} - \left(\int_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \right)^2$$

$$\begin{aligned} \text{Var}_{\mathbf{x} \sim q} \left[f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \right] &= \mathbb{E}_{\mathbf{x} \sim q} \left[\left(f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \right)^2 \right] - \left(\mathbb{E}_{\mathbf{x} \sim q} \left[f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \right] \right)^2 \\ &= \int_{\mathbf{x} \in \mathcal{X}} \left(f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \right)^2 q(\mathbf{x}) d\mathbf{x} - \left(\int_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x}) d\mathbf{x} \right)^2 \\ &= \int_{\mathbf{x} \in \mathcal{X}} \left(f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \right)^2 q(\mathbf{x}) d\mathbf{x} - \left(\int_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \right)^2 \\ &= \int_{\mathbf{x} \in \mathcal{X}} (f(\mathbf{x}))^2 \frac{p(\mathbf{x})}{q(\mathbf{x})} p(\mathbf{x}) d\mathbf{x} - \left(\int_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \right)^2 \end{aligned}$$

if p and q follow any two different distributions, for any $\mathbf{x} \in \mathcal{X}$, there always exist:

$$\frac{p(\mathbf{x})}{q(\mathbf{x})} \neq 1$$

therefore:

$$\text{Var}_{\mathbf{x} \sim p} [f(\mathbf{x})] \neq \text{Var}_{\mathbf{x} \sim q} \left[f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} \right]$$

- practical variance: **different**

Summary: **When using the importance sampling for sampling, if the two probability distributions p and q are very different, the practical effect will be poor. Therefore, in order to ensure the relatively practical good results when using Importance Sampling, it is necessary to limit the two probability distributions p and q will not be much different.**

Trust Region Methods Policy gradient estimates can have high variance and algorithms can be sensitive to the settings of their hyperparameters. Several approaches have been proposed to make policy gradient algorithms more robust. One effective measure is to employ a trust region constraint that restricts the amount by which any update is allowed to change the policy. A popular algorithm that makes use of this idea is [trust region policy optimization\(TRPO\)](#)[?]. In every iteration given current parameters θ_{old} , TRPO collects a (relatively large) batch of data and optimizes the surrogate objective:

$$J_{\text{TRPO}}(\theta) = \mathbb{E}_{\tau \sim \rho(\tau; \mu, \pi_{\theta_{\text{old}}}, \mathcal{P})} \left[\sum_{t=0}^{T(\tau)-1} \gamma^t \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} A^{\pi_{\theta_{\text{old}}}}(s_t, a_t) \right]$$

subject to a constraint on how much the policy is allowed to change, expressed in terms of the Kullback-Leibler divergence(KL-divergence):

$$\mathbb{E}_{s \sim \pi_{\theta_{\text{old}}}} \left[D_{\text{KL}} \left(\pi_{\theta_{\text{old}}}(\cdot | s) \parallel \pi_{\theta}(\cdot | s) \right) \right] \leq \delta$$

where A^{π} is the advantage function given as:

$$A^{\pi_{\theta}}(s_t, a_t) = Q^{\pi_{\theta}}(s_t, a_t) - V^{\pi_{\theta}}(s_t)$$

25.4.2 Model formulation

Objective The goal of the reinforcement learning is to choose a policy π with parameters θ to maximize the expected return, and here we transform the on-policy policy gradient optimization into off-policy by importance sampling, the derivation process of objective function is as follow:

$$\begin{aligned} & \max_{\pi} \mathbb{E}_{s_0, a_0, s_1, \dots} \left[R_0 = \sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi, \mathcal{P}, \mu \right] \\ \Rightarrow & \max_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta}, \mathcal{P}, \mu)} \left[R(\tau) = \sum_{t=0}^{T(\tau)-1} \gamma^t r_t^{\tau} \right] \quad (T(\tau) \text{ is the length of trajectory } \tau) \\ \Rightarrow & \max_{\theta} \int_{\tau \in \mathcal{T}} R(\tau) \rho(\tau; \pi_{\theta}, \mathcal{P}, \mu) d\tau \quad (\text{definition of mathematical expectation}) \\ \Rightarrow & \max_{\theta} \int_{\tau \in \mathcal{T}} R(\tau) \frac{\rho(\tau; \pi_{\theta}, \mathcal{P}, \mu)}{\rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu) d\tau \quad (\text{definition of importance sampling, } \forall \tau \in \mathcal{T}, \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu) \neq 0) \\ \Rightarrow & \max_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \frac{\rho(\tau; \pi_{\theta}, \mathcal{P}, \mu)}{\rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \right] \quad (\text{definition of mathematical expectation}) \end{aligned}$$

Here \mathcal{T} is the trajectory space, for each trajectory τ , the probability density function of trajectory $\rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)$ depends on there parameter components:

- $s_0^{\tau} \sim \mu(\cdot), \forall \tau \in \mathcal{T}$
- $a_t^{\tau} \sim \pi(\cdot | s_t^{\tau}; \theta_{\text{old}}), \forall \tau \in \mathcal{T}, \forall t \in \mathbb{N}_0$
- $s_{t+1}^{\tau} \sim \mathcal{P}(\cdot | s_t^{\tau}, a_t^{\tau}), \forall \tau \in \mathcal{T}, \forall t \in \mathbb{N}_0$

Considering the Markov property on state-action path $(s_0^{\tau}, a_0^{\tau}, s_1^{\tau}, a_1^{\tau}, s_2^{\tau}, \dots, s_{T(\tau)}^{\tau})$ with length $T(\tau) + 1$, which sampled from $\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)$, we can rewrite the probability density function of trajectory $\rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)$ as follow:

$$\begin{aligned} \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu) &= \mu(s_0^{\tau}) \cdot \pi(a_0^{\tau} | s_0^{\tau}; \theta_{\text{old}}) \cdot \mathcal{P}(s_1^{\tau} | s_0^{\tau}, a_0^{\tau}) \cdot \pi(a_1^{\tau} | s_1^{\tau}; \theta_{\text{old}}) \cdot \mathcal{P}(s_2^{\tau} | s_1^{\tau}, a_1^{\tau}) \cdot \dots \cdot \mathcal{P}(s_{T(\tau)}^{\tau} | s_{T(\tau)-1}^{\tau}, a_{T(\tau)-1}^{\tau}) \\ &= \mu(s_0^{\tau}) \cdot \prod_{t=0}^{T(\tau)-1} \pi(a_t^{\tau} | s_t^{\tau}; \theta_{\text{old}}) \cdot \prod_{t=0}^{T(\tau)-1} \mathcal{P}(s_{t+1}^{\tau} | s_t^{\tau}, a_t^{\tau}) \end{aligned}$$

we can rewrite the objective as follow:

$$\begin{aligned} & \max_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \frac{\rho(\tau; \pi_{\theta}, \mathcal{P}, \mu)}{\rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \right] \\ \Rightarrow & \max_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \frac{\cancel{\mu(s_0^{\tau})} \cdot \prod_{t=0}^{T(\tau)-1} \pi(a_t^{\tau} | s_t^{\tau}; \theta) \cdot \prod_{t=0}^{T(\tau)-1} \cancel{\mathcal{P}(s_{t+1}^{\tau} | s_t^{\tau}, a_t^{\tau})}}{\cancel{\mu(s_0^{\tau})} \cdot \prod_{t=0}^{T(\tau)-1} \pi(a_t^{\tau} | s_t^{\tau}; \theta_{\text{old}}) \cdot \prod_{t=0}^{T(\tau)-1} \cancel{\mathcal{P}(s_{t+1}^{\tau} | s_t^{\tau}, a_t^{\tau})}} \right] \\ \Rightarrow & \max_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \frac{\prod_{t=0}^{T(\tau)-1} \pi(a_t^{\tau} | s_t^{\tau}; \theta)}{\prod_{t=0}^{T(\tau)-1} \pi(a_t^{\tau} | s_t^{\tau}; \theta_{\text{old}})} \right] \\ \Rightarrow & \max_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \log \left(\frac{\prod_{t=0}^{T(\tau)-1} \pi(a_t^{\tau} | s_t^{\tau}; \theta)}{\prod_{t=0}^{T(\tau)-1} \pi(a_t^{\tau} | s_t^{\tau}; \theta_{\text{old}})} \right) \right] \end{aligned}$$

$$\begin{aligned}
&\Rightarrow \max_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \left(\log \left(\prod_{t=0}^{T(\tau)-1} \pi(a_t^\tau | s_t^\tau; \theta) \right) - \log \left(\prod_{t=0}^{T(\tau)-1} \pi(a_t^\tau | s_t^\tau; \theta_{\text{old}}) \right) \right) \right] \\
&\Rightarrow \max_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \left(\sum_{t=0}^{T(\tau)-1} \log \pi(a_t^\tau | s_t^\tau; \theta) - \sum_{t=0}^{T(\tau)-1} \log \pi(a_t^\tau | s_t^\tau; \theta_{\text{old}}) \right) \right] \\
&\Rightarrow \max_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \sum_{t=0}^{T(\tau)-1} \log (\pi(a_t^\tau | s_t^\tau; \theta) - \pi(a_t^\tau | s_t^\tau; \theta_{\text{old}})) \right] \\
&\Rightarrow \max_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \sum_{t=0}^{T(\tau)-1} \log \frac{\pi(a_t^\tau | s_t^\tau; \theta)}{\pi(a_t^\tau | s_t^\tau; \theta_{\text{old}})} \right]
\end{aligned}$$

Besides, consider the constraint that:

$$\mathbb{E}_{s \sim \pi_{\theta_{\text{old}}}} [D_{\text{KL}} (\pi_{\theta_{\text{old}}}(\cdot | s) \| \pi_{\theta}(\cdot | s))] \leq \delta$$

using a penalty instead of a constraint, we can convert the constrained problem(TRPO):

$$\begin{aligned}
\max_{\theta} & \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \sum_{t=0}^{T(\tau)-1} \log \frac{\pi(a_t^\tau | s_t^\tau; \theta)}{\pi(a_t^\tau | s_t^\tau; \theta_{\text{old}})} \right] \\
\text{s.t. } & \mathbb{E}_{s \sim \rho(s; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} [D_{\text{KL}} (\pi_{\theta_{\text{old}}}(\cdot | s) \| \pi_{\theta}(\cdot | s))] \leq \delta
\end{aligned}$$

into an unconstrained problem(PPO):

$$\max_{\theta} J_{\text{PPO}}(\theta) = \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \sum_{t=0}^{T(\tau)-1} \log \frac{\pi(a_t^\tau | s_t^\tau; \theta)}{\pi(a_t^\tau | s_t^\tau; \theta_{\text{old}})} \right] - \beta \mathbb{E}_{s \sim \rho(s; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} [D_{\text{KL}} (\pi_{\theta_{\text{old}}}(\cdot | s) \| \pi_{\theta}(\cdot | s))]$$

where the hyperparameter $\beta > 0$ is a penalty coefficient.

Convert maximize log-likelihood to minimize kl-divergence for stochastic policy Here we consider the stochastic policy case, each action a is sampled from its corresponding known action probability distribution \mathbf{p} (e.g. $a = 2$ is sampled from its corresponding action probability distribution $\mathbf{p} = (0.02, 0.01, 0.92, 0.05)$):

$$a \in \mathcal{A} \sim \mathbf{p} \in \mathbb{P}^{|\mathcal{A}|}$$

therefore we can convert the **first term** of above objective with maximizing log-likelihood to minimizing kl-divergence:

$$\begin{aligned}
\max_{\theta} & \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \sum_{t=0}^{T(\tau)-1} \left(\log \pi(a_t^\tau | s_t^\tau; \theta) - \log \pi(a_t^\tau | s_t^\tau; \theta_{\text{old}}) \right) \right] \\
\Rightarrow \min_{\theta} & \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \sum_{t=0}^{T(\tau)-1} \left(D_{\text{KL}} (\mathbf{p}_t^\tau \| \pi(\cdot | s_t^\tau; \theta)) - D_{\text{KL}} (\mathbf{p}_t^\tau \| \pi(\cdot | s_t^\tau; \theta_{\text{old}})) \right) \right] \\
\Rightarrow \min_{\theta} & \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \sum_{t=0}^{T(\tau)-1} \left(\sum_{k \in \mathcal{A}} p_{t,k}^\tau \log \frac{p_{t,k}^\tau}{\pi_{t,k}^\tau(\theta)} - \sum_{k \in \mathcal{A}} p_{t,k}^\tau \log \frac{p_{t,k}^\tau}{\pi_{t,k}^\tau(\theta_{\text{old}})} \right) \right]
\end{aligned}$$

$$\begin{aligned} & \Rightarrow \min_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \sum_{t=0}^{T(\tau)-1} \sum_{k \in \mathcal{A}} p_{t,k}^{\tau} \log \frac{\frac{p_{t,k}^{\tau}(\theta)}{\pi_{t,k}^{\tau}(\theta)}}{\frac{p_{t,k}^{\tau}}{\pi_{t,k}^{\tau}(\theta_{\text{old}})}} \right] \\ & \Rightarrow \min_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \sum_{t=0}^{T(\tau)-1} \sum_{k \in \mathcal{A}} p_{t,k}^{\tau} \log \frac{\pi_{t,k}^{\tau}(\theta_{\text{old}})}{\pi_{t,k}^{\tau}(\theta)} \right] \end{aligned}$$

also we can convert the second term of objective as follow in order to matching the first term:

$$\begin{aligned} & \max_{\theta} -\beta \mathbb{E}_{s \sim \rho(s; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[D_{\text{KL}} \left(\pi_{\theta_{\text{old}}}(\cdot | s) \middle\| \pi_{\theta}(\cdot | s) \right) \right] (\beta > 0) \\ & \Rightarrow \min_{\theta} \beta \mathbb{E}_{s \sim \rho(s; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[D_{\text{KL}} \left(\pi_{\theta_{\text{old}}}(\cdot | s) \middle\| \pi_{\theta}(\cdot | s) \right) \right] \\ & \Rightarrow \min_{\theta} \beta \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} D_{\text{KL}} \left(\pi_{\theta_{\text{old}}}(\cdot | s_t^{\tau}) \middle\| \pi_{\theta}(\cdot | s_t^{\tau}) \right) \right] \\ & \Rightarrow \min_{\theta} \beta \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} \sum_{k \in \mathcal{A}} \pi_{t,k}^{\tau}(\theta_{\text{old}}) \log \frac{\pi_{t,k}^{\tau}(\theta_{\text{old}})}{\pi_{t,k}^{\tau}(\theta)} \right] \end{aligned}$$

Therefore combine two terms of object, we can rewrite objective as follow:

$$\begin{aligned} & \min_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \sum_{t=0}^{T(\tau)-1} \sum_{k \in \mathcal{A}} p_{t,k}^{\tau} \log \frac{\pi_{t,k}^{\tau}(\theta_{\text{old}})}{\pi_{t,k}^{\tau}(\theta)} \right] + \beta \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} \sum_{k \in \mathcal{A}} \pi_{t,k}^{\tau}(\theta_{\text{old}}) \log \frac{\pi_{t,k}^{\tau}(\theta_{\text{old}})}{\pi_{t,k}^{\tau}(\theta)} \right] \\ & \Rightarrow \min_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[R(\tau) \sum_{t=0}^{T(\tau)-1} \sum_{k \in \mathcal{A}} p_{t,k}^{\tau} \log \frac{\pi_{t,k}^{\tau}(\theta_{\text{old}})}{\pi_{t,k}^{\tau}(\theta)} + \beta \sum_{t=0}^{T(\tau)-1} \sum_{k \in \mathcal{A}} \pi_{t,k}^{\tau}(\theta_{\text{old}}) \log \frac{\pi_{t,k}^{\tau}(\theta_{\text{old}})}{\pi_{t,k}^{\tau}(\theta)} \right] \\ & \Rightarrow \min_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} \sum_{k \in \mathcal{A}} \left(p_{t,k}^{\tau} \log \frac{\pi_{t,k}^{\tau}(\theta_{\text{old}})}{\pi_{t,k}^{\tau}(\theta)} R(\tau) + \beta \pi_{t,k}^{\tau}(\theta_{\text{old}}) \log \frac{\pi_{t,k}^{\tau}(\theta_{\text{old}})}{\pi_{t,k}^{\tau}(\theta)} \right) \right] \\ & \Rightarrow \min_{\theta} \mathbb{E}_{\tau \sim \rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)} \left[\sum_{t=0}^{T(\tau)-1} \sum_{k \in \mathcal{A}} \left(\underbrace{-p_{t,k}^{\tau} \log \pi_{t,k}^{\tau}(\theta)}_{\text{cross-entropy}} R(\tau) + \underbrace{\beta \pi_{t,k}^{\tau}(\theta_{\text{old}}) \log \frac{\pi_{t,k}^{\tau}(\theta_{\text{old}})}{\pi_{t,k}^{\tau}(\theta)}}_{\text{kl-divergence}} \right) \right] \\ & \Rightarrow \min_{\theta} \int_{\tau \in \mathcal{T}} \left(\sum_{t=0}^{T(\tau)-1} \sum_{k \in \mathcal{A}} \left(-p_{t,k}^{\tau} \log \pi_{t,k}^{\tau}(\theta) R(\tau) + \beta \pi_{t,k}^{\tau}(\theta_{\text{old}}) \log \frac{\pi_{t,k}^{\tau}(\theta_{\text{old}})}{\pi_{t,k}^{\tau}(\theta)} \right) \right) \underbrace{\rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)}_{\text{constant}} d\tau \\ & \Rightarrow \min_{\theta} \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} \sum_{k \in \mathcal{A}} \left(-p_{t,k}^{\tau^{(i)}} \log \pi_{t,k}^{\tau^{(i)}}(\theta) R(\tau^{(i)}) + \beta \pi_{t,k}^{\tau^{(i)}}(\theta_{\text{old}}) \log \frac{\pi_{t,k}^{\tau^{(i)}}(\theta_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\theta)} \right) \end{aligned}$$

where N is the number of trajectories sampled under $\rho(\tau; \pi_{\theta_{\text{old}}}, \mathcal{P}, \mu)$.

Variance reduction by reward-to-go and introducing baseline

$$\begin{aligned} & \min_{\theta} \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} \sum_{k \in \mathcal{A}} \left(-p_{t,k}^{\tau^{(i)}} \log \pi_{t,k}^{\tau^{(i)}}(\theta) R(\tau^{(i)}) + \beta \pi_{t,k}^{\tau^{(i)}}(\theta_{\text{old}}) \log \frac{\pi_{t,k}^{\tau^{(i)}}(\theta_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\theta)} \right) \\ & \Rightarrow \min_{\theta} \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} \sum_{k \in \mathcal{A}} \left(-p_{t,k}^{\tau^{(i)}} \log \pi_{t,k}^{\tau^{(i)}}(\theta) \left(\sum_{t'=t}^{T(\tau^{(i)})} \gamma^{t'-t} r_{t'}^{\tau^{(i)}} \right) + \beta \pi_{t,k}^{\tau^{(i)}}(\theta_{\text{old}}) \log \frac{\pi_{t,k}^{\tau^{(i)}}(\theta_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\theta)} \right) \end{aligned}$$

$$\Rightarrow \min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} \sum_{k \in \mathcal{A}} \left(-p_{t,k}^{\tau^{(i)}} \log \pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}) \left(\sum_{t'=t}^{T(\tau^{(i)})} \gamma^{t'-t} r_{t'}^{\tau^{(i)}} - b \right) + \beta \pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}}) \log \frac{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})} \right)$$

Tricks

- **Adaptive penalty:** Dynamic adjustment of penalty coefficient, i.e.,:

$$\begin{aligned} \frac{1}{N \times T} \sum_{i=0}^{N-1} \sum_{t=0}^{T-1} \left(\sum_{k \in \mathcal{A}} \pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}}) \log \frac{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})} \right) > \alpha_{\text{high}} \cdot D_{\text{KL-target}} &\implies \beta \leftarrow 2 \times \beta \\ \frac{1}{N \times T} \sum_{i=0}^{N-1} \sum_{t=0}^{T-1} \left(\sum_{k \in \mathcal{A}} \pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}}) \log \frac{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})} \right) < \alpha_{\text{low}} \cdot D_{\text{KL-target}} &\implies \beta \leftarrow \frac{\beta}{2} \end{aligned}$$

where $\alpha_{\text{high}} > 1$, $\alpha_{\text{low}} \in (0, 1)$ and $D_{\text{KL-target}}$ are all hyperparameters.

- **Clipping operation:**

$$\frac{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})} \Rightarrow \text{clip} \left(\frac{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})}; 1 - \epsilon, 1 + \epsilon \right)$$

Algorithm 36: Proximal Policy Optimization[?]

Input: N : the number of trajectories collected in each policy iteration;
 M : the number of batch update in each policy iteration;
 T : the length of sampled trajectory.

Input: θ_0 : initial policy parameters;
 ψ_0 : initial value function parameters.

Output: the final stochastic policy parameters: θ .

1 **for** $k = 0, 1, 2, \dots$ **do**

2 Collect set of trajectories $\mathcal{D}_k = \{\tau\}$ by running policy π_{θ_k} in the environment.

3 Compute reward-to-go:

$$\hat{R}_t^\tau = \sum_{t'=t}^T \gamma^{t'-t} r(s_{t'}, a_{t'}, s_{t'+1}^\tau), \forall \tau \in \mathcal{D}_k, t \in \{0, \dots, T\}$$

4 Compute advantage estimates, \hat{A}_t^τ (using any method of advantage estimation) based on the current value function $V^{\pi_{\theta_k}}(s; \psi_k)$, for example:

$$\hat{A}^{\pi_{\theta_k}}(s_t^\tau, a_t^\tau) = r(s_t^\tau, a_t^\tau, s_{t+1}^\tau) + \gamma V^{\pi_{\theta_k}}(s_{t+1}^\tau; \psi_k) - V^{\pi_{\theta_k}}(s_t^\tau; \psi_k), \forall \tau \in \mathcal{D}_k, t \in \{0, \dots, T\}$$

5 Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left(\frac{\pi_{\theta}(a_t^\tau | s_t^\tau)}{\pi_{\theta_k}(a_t^\tau | s_t^\tau)} \hat{A}^{\pi_{\theta_k}}(s_t^\tau, a_t^\tau), g(\epsilon, \hat{A}^{\pi_{\theta_k}}(s_t^\tau, a_t^\tau)) \right)$$

6 Fit value function by regression on mean-squared error:

$$\psi_{k+1} = \arg \min_{\psi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left(V_{\psi}(s_t^\tau) - \hat{R}_t^\tau \right)^2$$

7 **end**

- As the iterative step k proceed:
 - the more accurate the estimation of the value function $V^{\pi_{\theta}}(s; \psi)$,
 - the more accurate the estimation of the advantage function $\hat{A}^{\pi_{\theta}}(s, a) = r + \gamma V^{\pi_{\theta}}(s'; \psi) - V^{\pi_{\theta}}(s; \psi)$,
 - the more accurate updates of stochastic policy π_{θ} .
- Training supervised label \hat{R}_t^τ corresponding to $V(s; \psi)$ comes from Monte Carlo sampling.

Problem: How we get PPO-Clip objective in above pseudo-code?

25.4.3 Reference

- Jonathan Hui: RL—Importance Sampling
- Proximal Policy Optimization Algorithms - PPO

25.5 Asynchronous Methods for Deep Reinforcement Learning (A3C)

25.5.1 Introduction

本节选自论文Asynchronous Methods for Deep Reinforcement Learning[?]

25.6 Sample Efficient Actor-Critic with Experience Replay (ACER)

25.6.1 Introduction

本节选自论文Sample Efficient Actor-Critic with Experience Replay[?]

25.7 Deterministic Policy Gradient Algorithms (DPG)

25.7.1 Introduction

本节选自Deterministic Policy Gradient Algorithms[?]

25.8 Continuous Control with Deep Reinforcement Learning (DDPG)

25.8.1 Introduction

本节选自论文Continuous control with deep reinforcement learning[?]

Quick Facts

- DDPG is an off-policy algorithm.
- DDPG can only be used for environments with continuous action spaces.
- DDPG can be thought of as being deep Q -learning for continuous action spaces.

25.8.2 Model formulation

The Q-learning side of DDPG

- Bellman equation in theory:

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim \mathcal{P}(\cdot | s, a)} \left[r(s, a, s') + \gamma \mathbb{E}_{a' \sim \pi(\cdot | s')} [Q^\pi(s', a')] \right]$$

- Bellman equation in practical as target in discrete action space:

$$y = r(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta')$$

using max operation instead expectation.

- Bellman equation in practical as target in continuous action space:

$$y = r(s, a, s') + \gamma Q(s', \mu(s'; \theta); \phi)$$

using policy directly.

Note: 因为在 discrete action space 环境下使用 max 操作代替期望，本身就是不准确的，因此在 continuous action space 环境下，直接使用策略也是完全没有问题的，毕竟大家都是不准确的，都认为近似等于期望

Algorithm 37: Deep Deterministic Policy Gradient(DDPG)[?]

Input: θ : initial policy parameters;
 ϕ : initial Q -function parameters;
 \mathcal{D} : empty replay buffer.

Output: final deterministic policy $\mu_\theta : \mathcal{S} \rightarrow \mathcal{A}$

1 Set target parameters equal to primary parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$

2 **repeat**

3 Observe state s and select an action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{\text{low}}, a_{\text{high}})$, where $\epsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$ for action exploration

4 Execute a in the environment

5 Observe next state s' , reward r , and done signal d to indicate whether s' is terminal

6 Store (s, a, s', r, d) in replay buffer \mathcal{D}

7 If s' is terminal, reset environment state

8 **if** it's time to update **then**

9 **for** however many updates **do**

10 Randomly sample a batch of transitions $\mathcal{B} = \{(s, a, s', r, d)\}$ from \mathcal{D}

11 Compute Q-targets for all transitions in \mathcal{B} :

$$y(s', r, d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

12 Update Q-function by one step of gradient descent using:

$$\nabla_\phi \frac{1}{|\mathcal{B}|} \sum_{(s, a, s', r, d) \in \mathcal{B}} (Q_\phi(s, a) - y(s', r, d))^2$$

13 Update policy by one step of gradient ascent using:

$$\nabla_\theta \frac{1}{|\mathcal{B}|} \sum_{s \in \mathcal{B}} Q_\phi(s, \mu_\theta(s))$$

14 Update target networks with:

$$\phi_{\text{targ}} \leftarrow \tau\phi + (1 - \tau)\phi_{\text{targ}}$$

$$\theta_{\text{targ}} \leftarrow \tau\theta + (1 - \tau)\theta_{\text{targ}}$$

15 **end**

16 **end**

17 **until** convergence;

Note:

- 随着学习次数越来越多， Q 函数的学习越来越准确，即：作为 critic，其越来越能够准确地对 state-action 对儿进行打分评判
- 在 policy 的学习过程中，我们假定当前 Q_θ 是公正准确的，那么在当前 state s 下，我们要做的是调

整策略参数 θ ，使得策略根据当前 state s 得到的输出 action $a = \mu_\theta(s)$ ，与 s 成对儿，被 Q_θ 评价打的分值要越高越好，即训练策略的优化目标是：

$$\max_{\theta} \frac{1}{|\mathcal{B}|} \sum_{s \in \mathcal{B}} Q_\phi(s, \mu_\theta(s))$$

- Actor $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$ ，其中 \mathcal{A} 一般为高维的特征向量
- Critic $Q_\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ ，其中神经网络中将 state 的特征向量和 action 的特征向量 concat，训练一个回归函数，训练中，网络输入 (s, a) 对应的监督标签为：

$$y(s', r, d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

- 在 Actor-Critic 体系下，Actor 对应的策略函数 π_θ 才是最终渴望得到，并且用于实际使用的。训练 Critic 只是为了更好地评价，并引导 π_θ 的学习，**Actor-Critic 类似 GAN 的思想，其中 Actor 对应 Generator，Critic 对应 Discriminator.**

25.8.3 Reference

- Patrick Emami: Deep Deterministic Policy Gradient in TensorFlow
- spinningup.openai: Deep Deterministic Policy Gradient
- krishnan: TensorFlow DDPG

25.9 Distributed Distributional Deterministic Policy Gradients (D4PG)

25.9.1 Introduction

本节选自论文Distributed Distributional Deterministic Policy Gradients[?]

25.10 Addressing Function Approximation Error in Actor-Critic Methods (TD3)

25.10.1 Introduction

本节选自论文Addressing Function Approximation Error in Actor-Critic Methods[?]

Addressing Bias

Addressing Variance

25.10.2 Model formulation

Algorithm 38: Twin Delayed DDPG(TD3)[?]

Input: θ : initial policy parameters; ϕ_1, ϕ_2 : initial Q-network parameters;

\mathcal{D} : empty replay buffer.

Output: final deterministic policy $\pi_\theta : \mathcal{S} \times \mathcal{A}$

```

1 Set target networks equal to primary networks:  $Q_{\phi'_1} \leftarrow Q_{\phi_1}$ ,  $Q_{\phi'_2} \leftarrow Q_{\phi_2}$ ,  $\mu_{\theta'} \leftarrow \mu_\theta$ 
2 repeat
3   Observe state  $s$  and select action with exploration noise  $a = \text{clip}(\mu_\theta(s) + \epsilon, \alpha_{\text{low}}, \alpha_{\text{high}})$ ,  

     $\epsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$ 
4   Execute  $a$  in the environment
5   Observe next state  $s'$ , immediate reward  $r$ , and done signal  $d$  to indicate whether  $s'$  is terminal
6   Store transition  $(s, a, s', r, d)$  in replay buffer  $\mathcal{D}$ 
7   If  $s'$  is terminal, reset environment state
8   if it's time to update then
9     for  $j$  in range(however many updates) do
10    Randomly sample mini-batch of transitions  $\mathcal{B} = \{(s, a, s', r, d)\}$  from  $\mathcal{D}$ 
11    Select next action in target actor network:  

12       $\tilde{a} \leftarrow \text{clip}(\mu_{\theta'}(s') + \text{clip}(\epsilon, -c, c), \alpha_{\text{low}}, \alpha_{\text{high}})$ ,  $\epsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$ ,  $\forall s' \in \mathcal{B}$ 
13    Compute target for two primary critic networks:  

14       $y \leftarrow r + \gamma(1 - d) \min_{i=1,2} Q_{\phi'_i}(s', \tilde{a})$ ,  $\forall (s', r, d, \tilde{a}) \in \mathcal{B}$ 
15    Update critics by one step gradient descent using:  

16       $\phi_i \leftarrow \arg \min_{\phi_i} \frac{1}{|\mathcal{B}|} \sum_{(s,a) \in \mathcal{B}} (y - Q_{\phi_i}(s, a))^2$ ,  $\forall i \in \{1, 2\}$ 
17    if  $j \bmod \text{policy-delay} = 0$  then
18      Update actor by one step gradient ascent using:  

19         $\theta \leftarrow \arg \max_{\theta} \frac{1}{|\mathcal{B}|} \sum_{s \in \mathcal{B}} Q_{\phi_1}(s, \mu_\theta(s))$ 
20      Update target networks with:  

21         $\phi'_i \leftarrow \tau \phi_i + (1 - \tau) \phi'_i$ ,  $\forall i \in \{1, 2\}$ 
22         $\theta' \leftarrow \tau \theta + (1 - \tau) \theta'$ 
23    end
24  end
25 until convergence;

```

25.10.3 Reference

- TD3: Learning To Run With AI

25.11 Soft Actor-Critic Algorithms and Applications (SAC) (未完成)

25.11.1 Introduction

本节选自论文Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor[?] 和Soft Actor-Critic Algorithms and Applications[?]，内容引用自OpenAI Spinning Up: Soft Actor Critic.

Entropy Let x be a random variable with probability mass or density function $p(x)$. The entropy \mathcal{H} of x is computed from its distribution $p(x)$ according to:

$$\mathcal{H}(p(x)) = \mathbb{E}_{x \sim p(x)} \underbrace{[-\log p(x)]}_{\text{information}}$$

Entropy-Regularized Reinforcement Learning In entropy-regularized reinforcement learning, the agent gets a bonus reward at each time step proportional to the entropy of the policy at that timestep. This changes reinforcement learning problem to:

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\tau \sim \rho(\tau; \pi, \mathcal{P}, \mu)} \left[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t, s_{t+1}) + \alpha \mathcal{H}(\pi(\cdot | s_t))) \right]$$

where $\alpha > 0$ is the trade-off coefficient. (Note: we're assuming an infinite-horizon discounted setting here, and we'll do the same for the rest of this page.) $\mathcal{H}(\pi(\cdot | s_t))$ can be thought of as the “**amount of options**” **the policy has in each state, in expectation**. We can now define the slightly-different value functions in this setting. V^π is changed to include the entropy bonuses from every timestep:

$$V^\pi(s) = \mathbb{E}_{a_0, s_1, a_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t, s_{t+1}) + \alpha \mathcal{H}(\pi(\cdot | s_t))) \middle| s_0 = s \right]$$

Q^π is changed to include the entropy bonuses from every timestep **except the first timestep**:

$$Q^\pi(s, a) = \mathbb{E}_{s_1, a_1, s_2, \dots} \left[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t \mathcal{H}(\pi(\cdot | s_t)) \middle| s_0 = s, a_0 = a \right]$$

With these definitions, V^π and Q^π are connected by:

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot | s)} [Q^\pi(s, a)] + \alpha \mathcal{H}(\pi(\cdot | s))$$

Proof.

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{a_0, s_1, a_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t, s_{t+1}) + \alpha \mathcal{H}(\pi(\cdot | s_t))) \middle| s_0 = s \right] \\ &= \mathbb{E}_{a_0, s_1, a_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) + \alpha \sum_{t=0}^{\infty} \gamma^t \mathcal{H}(\pi(\cdot | s_t)) \middle| s_0 = s \right] \\ &= \mathbb{E}_{a \sim \pi(\cdot | s)} \mathbb{E}_{s_1, a_1, s_2, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) + \alpha \mathcal{H}(\cdot | s_0) + \alpha \sum_{t=1}^{\infty} \gamma^t \mathcal{H}(\pi(\cdot | s_t)) \middle| s_0 = s, a_0 = a \right] \\ &= \mathbb{E}_{a \sim \pi(\cdot | s)} \underbrace{\mathbb{E}_{s_1, a_1, s_2, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t \mathcal{H}(\pi(\cdot | s_t)) \middle| s_0 = s, a_0 = a \right]}_{Q^\pi(s, a)} + \alpha \mathcal{H}(\cdot | s) \end{aligned}$$

$$= \mathbb{E}_{a \sim \pi(\cdot | s)} [Q^\pi(s, a)] + \alpha \mathcal{H}(\cdot | s)$$

and the Bellman equation for Q^π is:

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_{s_1, a_1, s_2, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t \mathcal{H}(\pi(\cdot | s_t)) \middle| s_0 = s, a_0 = a \right] \\ &= \mathbb{E}_{s_1 \sim \mathcal{P}(\cdot | s_0, a_0)} \left[\mathbb{E}_{a_1, s_2, a_2, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t \mathcal{H}(\pi(\cdot | s_t)) \middle| s_1 \right] \middle| s_0 = s, a_0 = a \right] \\ &= \mathbb{E}_{s_1 \sim \mathcal{P}(\cdot | s_0, a_0)} \left[\mathbb{E}_{a_1, s_2, a_2, \dots} \left[r(s_0, a_0, s_1) + \sum_{t=1}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t \mathcal{H}(\pi(\cdot | s_t)) \middle| s_1 \right] \middle| s_0 = s, a_0 = a \right] \\ &= \mathbb{E}_{s_1 \sim \mathcal{P}(\cdot | s_0, a_0)} \left[r(s_0, a_0, s_1) + \mathbb{E}_{a_1, s_2, a_2, \dots} \left[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t \mathcal{H}(\pi(\cdot | s_t)) \middle| s_1 \right] \middle| s_0 = s, a_0 = a \right] \\ &= \mathbb{E}_{s_1 \sim \mathcal{P}(\cdot | s_0, a_0)} \left[r(s_0, a_0, s_1) + \gamma \mathbb{E}_{a_1, s_2, a_2, \dots} \left[\sum_{k=0}^{\infty} \gamma^k r(s_{1+k}, a_{1+k}, s_{2+k}) + \alpha \sum_{k=0}^{\infty} \gamma^k \mathcal{H}(\pi(\cdot | s_{1+k})) \middle| s_1 \right] \middle| s_0 = s, a_0 = a \right] \\ &= \mathbb{E}_{s' \sim \mathcal{P}(\cdot | s, a)} [r(s, a, s') + \gamma V(s')] \end{aligned}$$

25.11.2 Model formulation

Soft Actor Critic SAC concurrently learns:

- a policy: π_θ
- two Q -functions: Q_{ϕ_1}, Q_{ϕ_2}
- a value function: V_ψ

Learning Q. The Q -functions are learned by MSBE(mean squared batch error) minimization, using a target value network to form the Bellman backups. They both use the same target, like in TD3, and have loss functions:

$$l(\phi_i; \mathcal{D}) = \mathbb{E}_{(s, a, s', r, d) \sim \mathcal{D}} \left[\left(Q_{\phi_i}(s, a) - (r + \gamma(1-d)V_{\bar{\psi}}(s')) \right)^2 \right]$$

The target value network, like the target networks in DDPG[?] and TD3[?], is obtained by polyak averaging the value network parameters over the course of training.

Learning V. The value function is learned by exploiting (a sample-based approximation of) the connection between V^π and Q^π . Before we go into the learning rule, let's first rewrite the connection equation between V^π and Q^π by using the definition of entropy to obtain:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{a \sim \pi(\cdot | s)} [Q^\pi(s, a)] + \alpha \mathcal{H}(\pi(\cdot | s)) \\ &= \mathbb{E}_{a \sim \pi(\cdot | s)} [Q^\pi(s, a)] + \alpha \mathbb{E}_{a \sim \pi(\cdot | s)} [-\log \pi(a | s)] \\ &= \mathbb{E}_{a \sim \pi(\cdot | s)} [Q^\pi(s, a) - \alpha \log \pi(a | s)] \\ &\approx Q^\pi(s, \tilde{a}) - \alpha \log \pi(\tilde{a} | s) \quad (\text{approximate the expectation by sampling from the policy, } \tilde{a} \sim \pi(\cdot | s).) \end{aligned}$$

SAC sets up a mean-squared-error loss for V_ψ based on this approximation. But what Q -value do we use? SAC uses clipped double- Q like TD3 for learning the value function, and **takes the minimum Q -value between the two approximators**. So the SAC loss for value function parameters is:

$$l(\psi; \mathcal{D}) = \mathbb{E}_{s \sim \mathcal{D}, \tilde{a} \sim \pi_\theta(\cdot | s)} \left[\left(V_\psi(s) - \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}) - \alpha \log \pi_\theta(\tilde{a} | s) \right) \right)^2 \right]$$

Importantly, we **do not use actions from the replay buffer here**: these actions are sampled fresh from the current version of the policy π_θ .

Learning the Policy (未完成) The policy should, in each state s , act to maximize the expected future return plus expected future entropy. That is, it should maximize $V^\pi(s)$, which we expand out (as before) into

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot | s)} [Q^\pi(s, a) - \alpha \log \pi(a | s) | s]$$

The way we optimize the policy makes use of the **reparameterization trick**

Exploration vs. Exploitation SAC trains a stochastic policy with entropy regularization, and explores in an on-policy way. The entropy regularization coefficient α explicitly controls the explore-exploit tradeoff, with higher α corresponding to more exploration, and lower α corresponding to more exploitation. The right coefficient (the one which leads to the stablest / highest-reward learning) may vary from environment to environment, and could require careful tuning.

idea. adaptive turning of α

Algorithm 39: Soft Actor-Critic

Input: θ : initial policy parameters;
 ϕ_1, ϕ_2 : initial Q -function parameters;
 ψ : initial value function parameters;
 \mathcal{D} : empty replay buffer.

1 Set target parameters equal to main parameters $\bar{\psi} \leftarrow \psi$

2 **repeat**

3 Observe a state s and select an action $a \sim \pi_\theta(\cdot | s)$

4 Execute action a in the environment

5 Observe next state s' , immediate reward r , and done signal d to indicate whether s' is terminal

6 Store current transition (s, a, s', r, d) into replay buffer \mathcal{D}

7 If s' is terminal, reset environment state.

8 **if** it's time to update **then**

9 **for** j in range(however many updates) **do**

10 Randomly sample a batch of transitions, $\mathcal{B} = \{(s, a, s', r, d)\}$ from \mathcal{D}

11 Compute targets for Q and V functions of each transition (s, a, s', r, d) in \mathcal{B} :

$$y_q(r, s', d) = r + \gamma(1 - d)V_{\bar{\psi}}(s')$$

$$y_v(s) = \min_{i=1,2} Q_{\phi_i}(s, \tilde{a}) - \alpha \log \pi_\theta(\tilde{a} | s), \quad \tilde{a} \sim \pi_\theta(\cdot | s)$$

12 Update Q -functions by one step of gradient descent using:

$$\nabla_{\phi_i} \frac{1}{|\mathcal{B}|} \sum_{(s, a, s', r, d) \in \mathcal{B}} (Q_{\phi_i}(s, a) - y_q(r, s', d))^2, \quad \forall i \in \{1, 2\}$$

13 Update V -function by one step of gradient descent using:

$$\nabla_{\psi} \frac{1}{|\mathcal{B}|} \sum_{s \in \mathcal{B}} (V_{\psi}(s) - y_v(s))^2$$

14 Update policy by one step of gradient ascent using:

$$\nabla_{\theta} \frac{1}{|\mathcal{B}|} \sum_{s \in \mathcal{B}} \left(Q_{\phi}(s, \tilde{a}_{\theta}(s)) - \alpha \log \pi_\theta(\tilde{a}_{\theta}(s) | s) \right)$$

15 where $\tilde{a}_{\theta}(s)$ is a sample from $\pi_\theta(\cdot | s)$ which is differentiable wrt θ via the reparameterization trick.

16 Update target value network with:

$$\bar{\psi} \leftarrow \rho \bar{\psi} + (1 - \rho) \psi$$

17 **end**

18 **end**

19 **until** convergence;

Question & Answer

1. Why use double Q -networks in the model?
2. What entropy regularization works for?

25.11.3 Understanding the Impact of Entropy on Policy Optimization

Based on paper: [Understanding the Impact of Entropy on Policy Optimization\[?\]](#)

25.11.4 Reference

- OpenAI Spinning Up: Soft Actor-Critic

25.12 Boosting Soft Actor-Critic: Emphasizing Recent Experience without Forgetting the Past

25.12.1 Introduction

本节选自论文Boosting Soft Actor-Critic: Emphasizing Recent Experience without Forgetting the Past[?]

Soft Actor-Critic Algorithm SAC[?][?] achieves state-of-the-art performance on a range of **continuous-action** benchmark tasks, outperforming prior on-policy and off-policy methods, including: TRPO[?], PPO[?][?], DDPG[?], TD3[?].

SAC tries to maximize the expected sum of rewards and the entropy of a policy π :

$$\max_{\pi} J(\pi) = \mathbb{E}_{\tau \sim \rho(\tau; \pi, \mathcal{P}, \mu)} \left[\sum_{t=0}^T \gamma^t \left(r(s_t^\tau, a_t^\tau, s_{t+1}^\tau) + \alpha H(\pi(\cdot | s_t^\tau)) \right) \right]$$

Here ρ_π is the state-action marginals of the trajectory distribution induced by π . The hyper-parameter α balances exploitation and exploration, and affects the stochasticity of the optimal policy.

SAC consists of five networks:

- A policy network π_θ with parameters θ takes in the state and outputs the mean and standard deviation of an action distribution(stochastic policy).
- A Q-network Q_{ϕ_1} with parameters ϕ_1 to estimate the value of state-action pairs.
- A Q-network Q_{ϕ_2} with parameters ϕ_2 to estimate the value of state-action pairs.
- A state value network V_ψ with parameters ψ that estimates the value of a state.
- A target state value network $V_{\bar{\psi}}$ with parameters $\bar{\psi}$ which is simply an exponentially moving average of the state value network V_ψ .

Soft Actor-Critic with Emphasizing Recent Experience (SAC + ERE)

- **Core idea:** during the parameter update phase, the first mini-batch is sampled from all the data in the replay buffer, then for each subsequent minibatch we gradually reduce our range of sampling to **sample more aggressively from more recent data points**.
- **Two key points of this scheme:**
 1. sample more recent data with higher frequency.
 2. updates with older data do not overwrite the updates with the fresher data.
- **Methodology**
 - assume that in the current update phase we are going to make K times mini-batch updates;
 - N is the max size of the replay buffer;
 - for the k -th update, $k \in \{1, \dots, K\}$, sample **uniformly** from **the most recent c_k data points(transitions)**, where:

$$c_k = \max\{N \cdot \eta^{\frac{k}{K}1000}, c_{\min}\}$$

- $\eta \in (0, 1]$ is a hyperparameter that determines how much emphasis we put on recent data.
 - * when $\eta = 1$, $c_k = \max\{N, c_{\min}\} = N$, this is equivalent to uniform sampling;
 - * when $\eta < 1$, c_k decreases as we perform each update.
- c_{\min} as the minimum allowable value of c_k , which can help prevent **sampling from a very small amount of recent data, which may cause overfitting**.

Soft Actor-Critic with Prioritized Experience Replay (SAC + PER)

- Since SAC has two Q-networks, we redefine the absolute TD error $|\delta|$ of transition (s, a, s', r, d) to be the average absolute TD error of two Q-networks:

$$|\delta| = \frac{1}{2} \sum_{l=1}^2 |r + \gamma V_{\bar{\psi}}(s') - Q_{\phi_l}(s, a)|$$

within the sum, the first two terms $r + \gamma V_{\bar{\psi}}(s')$ is simply the target for the Q-network, and the third term $Q_{\phi_l}(s, a)$ is the current estimate of the l^{th} Q-network.

- For the i^{th} data point (transition in replay buffer), the definition of the priority value p_i is:

$$p_i = |\delta_i| + \epsilon$$

where ϵ is a very small constant (e.g. $\epsilon = 1e-6$).

- The probability of sampling the i^{th} data point $P(i)$ is computed as:

$$P(i) = \frac{p_i^{\beta_1}}{\sum_{j=1}^{|\mathcal{D}|} p_j^{\beta_1}}$$

where β_1 is a hyperparameter that controls how much the priority value affects the sampling probability, which is denoted by α in paper **Prioritized Experience Replay**[?].

- The importance sampling (IS) weight w_i for the i^{th} data point is computed as:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^{\beta_2}$$

where β_2 is denoted as β in paper **Prioritized Experience Replay**[?].

Problem: Why using importance sampling (IS) weight?

25.12.2 Model formulation (未完成)

Algorithm 40: Soft Actor Critic with Emphasizing Recent Experience and Prioritized Experience Replay (SAC + ERE + PER)

Input: θ : initial policy network parameters;

ϕ : initial Q-network parameters;

ψ : initial value network parameters;

\mathcal{D} : empty replay buffer

Output: final stochastic policy $\pi_\theta : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{P}$

```

1 Set target value network equal to primary value network:  $\bar{\psi} \leftarrow \psi$ 
2 repeat
3   Observe state  $s$  and select action  $a \sim \pi_\theta(\cdot | s)$ 
4   Execute  $a$  in the environment
5   Observe next state  $s'$ , immediate reward  $r$ , and done signal  $d$  to indicate whether  $s'$  is terminal
6   Store transition in replay buffer  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s, a, s', r, d)\}$ 
7   If  $s'$  is terminal, reset environment state
8   if it's time  $t$  to update then
9     Compute the annealed  $\eta$ :  $\eta_t = \eta_0 + (\eta_T - \eta_0) \cdot \frac{t}{T}$ 
10     $t \leftarrow t + 1$ 
11    for  $k = 1, \dots, K$  mini-batch update do
12      Compute the sampling range inside  $\mathcal{D}$ :  $c_k = |\mathcal{D}| \cdot \eta_t^{\frac{k}{K} \times 1000}$ 
13      Sample a mini-batch transitions with probabilities  $P(i)$ :  $\mathcal{B} \sim \mathcal{D}_{c_k}$ ,  $P(i) = \frac{p_i^{\beta_1}}{\sum_{j \in \mathcal{D}_{c_k}} p_j^{\beta_1}}$ 
14      Set  $\Delta\psi, \Delta\phi, \Delta\theta = 0$ 
15      for each data point  $b \in \mathcal{B}$  do
16        Compute importance sampling weight:  $w_b = \frac{(\frac{1}{N} \cdot \frac{1}{P(b)})^{\beta_2}}{\max_{j \in \mathcal{B}} w_j}$ 
17        Get abs TD-error:  $|\delta_b| = \frac{1}{2} \sum_{l=1}^2 |r + \gamma V_{\bar{\psi}}(s') - Q_{\phi_l}(s, a)|$ 
18        Update priority:  $p_b \leftarrow |\delta_b| + \epsilon$ 
19        Accumulate weight-change of  $\psi$  computed on transition  $b$ :  $\Delta\psi \leftarrow \Delta\psi + w_b \nabla$ 
20      end
21    end
22  end
23 until convergence;

```

25.13 Action Robust Reinforcement Learning and Applications in Continuous Control

25.13.1 Introduction

本节选自论文Action Robust Reinforcement Learning and Applications in Continuous Control[?]

25.14 Remember and Forget for Experience Replay (ReF-ER)

25.14.1 Introduction

本节选自论文Remember and Forget for Experience Replay[?]

25.14.2 Model formulation

stochastic policy + ReF-ER

deterministic policy + ReF-ER

25.15 Dimension-Wise Importance Sampling Weight Clipping for Sample-Efficient Reinforcement Learning

25.15.1 Introduction

本节选自论文Dimension-Wise Importance Sampling Weight Clipping for Sample-Efficient Reinforcement Learning[?]

26 Value-Based Deep Reinforcement Learning Algorithms

26.1 Deep Q-Network (DQN)

26.1.1 Introduction

本节选自 DeepMind 论文 Playing Atari with Deep Reinforcement Learning[?] 和 Nature 版 Human-level control through deep reinforcement learning[?]

Q-learning

- State-action value function(Q function):

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t+2}, \dots} \left[R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t, a_t; \pi, \mathcal{P} \right]$$

- Bellman equation

$$\begin{aligned} Q^\pi(s_t, a_t) &= \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t+2}, \dots} \left[R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t, a_t; \pi, \mathcal{P} \right] \\ &= \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t+2}, \dots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \mid s_t, a_t; \pi, \mathcal{P} \right] \\ &= \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t+2}, \dots} \left[r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid s_t, a_t; \pi, \mathcal{P} \right] \\ &= \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} [r_t] + \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t+2}, \dots} \left[\gamma \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid s_t, a_t; \pi, \mathcal{P} \right] \\ &= \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} [r_t] + \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} \mathbb{E}_{a_{t+1} \sim \pi(\cdot | s_{t+1})} \mathbb{E}_{s_{t+2}, a_{t+2}, s_{t+3}, \dots} \left[\gamma \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid s_{t+1}, a_{t+1}; \pi, \mathcal{P} \right] \\ &= \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} \left[r_t + \mathbb{E}_{a_{t+1} \sim \pi(\cdot | s_{t+1})} \underbrace{\gamma \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid s_{t+1}, a_{t+1}; \pi, \mathcal{P}}_{Q^\pi(s_{t+1}, a_{t+1})} \right] \\ &= \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} \left[r_t + \gamma \mathbb{E}_{a_{t+1} \sim \pi(\cdot | s_{t+1})} Q^\pi(s_{t+1}, a_{t+1}) \mid s_t, a_t; \pi, \mathcal{P} \right] \end{aligned}$$

- Temporal difference(TD) error

- if Q -value estimates are accurate, the following must hold:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} \left[r_t + \gamma \mathbb{E}_{a_{t+1} \sim \pi(\cdot | s_{t+1})} Q^\pi(s_{t+1}, a_{t+1}) \mid s_t, a_t; \pi, \mathcal{P} \right]$$

- if not, there is an error:

$$\delta = \underbrace{\mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} \left[r_t + \gamma \mathbb{E}_{a_{t+1} \sim \pi(\cdot | s_{t+1})} Q^\pi(s_{t+1}, a_{t+1}) \mid s_t, a_t; \pi, \mathcal{P} \right]}_{\text{target}} - \underbrace{Q^\pi(s_t, a_t)}_{\text{prediction}}$$

- to learn better Q -value estimates - minimize δ

Pseudo-code of Q-learning is as follow, note that in theory, we seem to update Q value using TD error as follow:

$$\delta = \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} \left[r(s_t, a_t, s_{t+1}) + \gamma \mathbb{E}_{a_{t+1} \sim \pi(\cdot | s_{t+1})} Q^\pi(s_{t+1}, a_{t+1}) \middle| s_t, a_t; \pi, \mathcal{P} \right] - Q^\pi(s_t, a_t)$$

but in practice, we update Q value using TD error as follow:

$$\delta = r(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t)$$

because of the usage of maximum value of Q over all actions of a_{t+1} , the target is set relatively higher than expected value, so we use a step size $\alpha \in (0, 1]$ to alleviate the problem. Therefore we update Q value as follow:

$$Q^\pi(s_t, a_t) \leftarrow (1 - \alpha)Q^\pi(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} Q^\pi(s_{t+1}, a_{t+1}) \right]$$

namely:

$$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t) \right]$$

The smaller the value of α , the more conservative the update is.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

 until S is terminal

图 128: Algorithm from: Sutton & Barto 2018, chapter 6, page 131[?]

Q-learning with function approximation To generalize over states and actions, parameterize Q with a function approximator, e.g. a deep neural network. The **TD error serves as loss**:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{s_t \in \mathcal{S}, a_t \in \mathcal{A}, s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)} \left[\underbrace{\left(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \boldsymbol{\theta}') - Q(s_t, a_t; \boldsymbol{\theta}) \right)^2}_{\text{target}} \right]$$

and is optimized using gradient descent.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{s \in \mathcal{S}, a \in \mathcal{A}, s' \sim \mathcal{P}(\cdot | s, a)} \left[\left(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \boldsymbol{\theta}') - Q(s, a; \boldsymbol{\theta}) \right) \nabla_{\boldsymbol{\theta}} Q(s, a; \boldsymbol{\theta}) \right]$$

26.1.2 Model formulation

Algorithm 41: Deep Q-learning with Experience Replay[?]

Input: N : capacity of replay memory \mathcal{D} ;

M : number of episodes for training;

T : max number of steps during each episode;

C : frequency for updating \hat{Q} during training.

Output: final parameters of Q function $\boldsymbol{\theta}$.

```

1 Initialize replay memory  $\mathcal{D}$  to capacity  $N$ 
2 Initialize action-value function  $Q$  with random weights  $\boldsymbol{\theta}$ 
3 Initialize target action-value function  $\hat{Q}$  with weights  $\boldsymbol{\theta}^- = \boldsymbol{\theta}$ 
4 for  $episode = 1, \dots, M$  do
5   Initialize sequence  $s_1 = \{\mathbf{x}_1\}$  and preprocessed observation  $\phi(\mathbf{x}_1)$ 
6   for  $t = 1, \dots, T$  do
7     Select action using  $\epsilon$ -greedy strategy:
      
$$a_t = \begin{cases} \text{random from } \mathcal{A}, & \text{with probability } \epsilon \in (0, 1) \\ \arg \max_{a \in \mathcal{A}} Q(\phi(s_t), a; \boldsymbol{\theta}), & \text{with probability } 1 - \epsilon \end{cases}$$

8     Execute action  $a_t$  in emulator and observe reward  $r_t$  and state  $\mathbf{x}_{t+1}$ 
9     Set  $s_{t+1} = s_t \cup \{a_t, \mathbf{x}_{t+1}\}$  and preprocess state to get observation  $\phi_{t+1} = \phi(\mathbf{x}_{t+1})$ 
10    Store transition  $(\phi_t, a_t, \phi_{t+1}, r_t)$  in  $\mathcal{D}$ 
11    Sample random minibatch of transitions  $\mathcal{U}(\mathcal{D}) = \{(\phi_j, a_j, \phi_{j+1}, r_j)\}_{j=1}^m$  from  $\mathcal{D}$ 
12    Set the target of state-action value function as follow:
      
$$y_j = \begin{cases} r_j, & \text{if episode terminates at step } j + 1 \\ r_j + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(\phi_{j+1}, a'; \boldsymbol{\theta}^-), & \text{otherwise} \end{cases}$$

13    Perform a gradient descent step on the loss function with respect to the network parameters
14     $\boldsymbol{\theta}$ :
      
$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{j=1}^m (y_j - Q(\phi_j, a_j; \boldsymbol{\theta}))^2$$

15    Every  $C$  steps reset  $\hat{Q} \leftarrow Q$ , namely reset  $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$ 
16  end
end
```

26.1.3 Reference

- Going Deeper Into Reinforcement Learning: Understanding Deep-Q-Networks
- Introduction to Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG)

26.2 Deep Reinforcement Learning with Double Q-learning (Double DQN)

26.2.1 Introduction

本节选自论文Deep Reinforcement Learning with Double Q-learning[?]

DQN \Rightarrow **Double DQN**

- The Q -target corresponding to predicted state-action value $Q(\mathbf{s}_t, a_t; \boldsymbol{\theta})$ is:

$$\begin{aligned} y_t^{\text{DQN}} &= Q(\mathbf{s}_t, a_t; \boldsymbol{\theta}) \\ &= \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{s}_t, a_t)} [r(\mathbf{s}_t, a_t, \mathbf{s}_{t+1}) + \gamma \mathbb{E}_{a_{t+1} \sim \pi(\cdot | \mathbf{s}_{t+1})} Q(\mathbf{s}_{t+1}, a_{t+1}; \boldsymbol{\theta}) | \mathbf{s}_t, a_t; \pi, \mathcal{P}] \\ &\approx r(\mathbf{s}_t, a_t, \mathbf{s}_{t+1}) + \gamma \max_{a \in \mathcal{A}} Q(\mathbf{s}_{t+1}, a; \boldsymbol{\theta}) \end{aligned}$$

The max operator in standard Q-learning and DQN, uses the same value both to select and to evaluation an action. This makes it more likely to select **overestimated** values, **resulting in overoptimistic value estimates**. To prevent this, we can **decouple the selection from the evaluation**.

- In Double Q-learning[?], two value functions are learned by assigning experiences randomly to update one of the two value functions, resulting in two sets of weights, $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$. For each update, one set of weights is used to determine the greedy policy and the other to determine its value.
- For a clear comparison, we can untangle the selection and evaluation in Q -learning and rewrite its target as:

$$\begin{aligned} y_t^{\text{DQN}} &= r(\mathbf{s}_t, a_t, \mathbf{s}_{t+1}) + \gamma \max_{a \in \mathcal{A}} Q(\mathbf{s}_{t+1}, a; \boldsymbol{\theta}) \\ &= r(\mathbf{s}_t, a_t, \mathbf{s}_{t+1}) + \underbrace{\gamma Q(\mathbf{s}_{t+1}, \underbrace{\arg \max_{a \in \mathcal{A}} Q(\mathbf{s}_{t+1}, a; \boldsymbol{\theta})}_{\text{selection}}; \boldsymbol{\theta})}_{\text{evaluation}} \end{aligned}$$

- The Q -target of Double Q-learning can be written as:

$$y_t^{\text{DoubleDQN}} = r(\mathbf{s}_t, a_t, \mathbf{s}_{t+1}) + \underbrace{\gamma Q(\mathbf{s}_{t+1}, \underbrace{\arg \max_{a \in \mathcal{A}} Q(\mathbf{s}_{t+1}, a; \boldsymbol{\theta}')}_{\text{selection}}; \boldsymbol{\theta}')}_{\text{evaluation}}$$

Notice that the **selection of the action**, in the argmax, is still due to the **online weights $\boldsymbol{\theta}$** . This means that, as in Qlearning, we are still estimating the value of the greedy policy according to the current values, as defined by $\boldsymbol{\theta}$. However, we use the second set of weights $\boldsymbol{\theta}'$ to fairly evaluate the value of this policy.

26.2.2 Model formulation

Algorithm 42: Double DQN Algorithm[?]

Input: \mathcal{D} : empty replay buffer;

\mathcal{B} : training batch buffer;

θ : initial network parameters;

θ^- : copy of θ ;

N_r : replay buffer maximum size;

N_f : frame sequence maximum size;

N^- : target network replacement frequency.

```

1 for episode  $e \in \{1, \dots, M\}$  do
2   Initialize frame sequence  $\mathbf{x} \leftarrow ()$ 
3   for  $t \in \{0, 1, \dots\}$  do
4     Get the state  $s = \phi(\mathbf{x})$ ,  $\phi$  is preprocessing function mapping from frame sequence to state.
5     Sample action using  $\epsilon$ -greedy strategy:
6       
$$a = \begin{cases} \text{random from } \mathcal{A}, & \text{with probability } \epsilon \in (0, 1) \\ \arg \max_{a \in \mathcal{A}} Q(s, a; \theta), & \text{with probability } 1 - \epsilon \end{cases}$$

7     Execute action  $a$  in emulator, observe reward  $r$ , next frame  $\mathbf{x}_{t+1}$ , also append  $\mathbf{x}_{t+1}$  to  $\mathbf{x}$ .
8     if  $|\mathbf{x}| > N_f$  then
9       | delete the oldest frame  $\mathbf{x}_{t_{\min}}$  from frame sequence  $\mathbf{x}$ 
10    Set  $s' = \phi(\mathbf{x})$ , and add transition tuple  $(s, a, s', r)$  to replay buffer  $\mathcal{D}$ 
11    if  $|\mathcal{D}| > N_r$  then
12      | replace the oldest tuple in replay buffer  $\mathcal{D}$ 
13    end
14    Sample a minibatch  $\mathcal{B}$  of tuples  $(s, a, s', r) \sim \text{Uniform}(\mathcal{D})$ 
15    Construct target values, one for each of the  $N_b$  tuples:
16      
$$y^{\text{DoubleDQN}} = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma Q(s', a^{\max}(s'; \theta); \theta^-) & \text{otherwise.} \end{cases}$$

17      where:  $a^{\max}(s'; \theta) = \arg \max_{a' \in \mathcal{A}} Q(s', a'; \theta)$ 
18      Perform a gradient descent step on the loss function with respect to the network parameters
19       $\theta$ :
20        
$$J(\theta) = \frac{1}{|\mathcal{B}|} \sum_{(s, a, s', r) \in \mathcal{B}} (y^{\text{DoubleDQN}} - Q(s, a; \theta))^2$$

21      Replace target parameters  $\theta^- \leftarrow \theta$  every  $N^-$  steps.
22    end
23 end

```

26.3 Dueling Network Architectures for Deep Reinforcement Learning (Dueling DQN)

26.3.1 Introduction

本节选自论文Dueling Network Architectures for Deep Reinforcement Learning[?]

Decomposition of the Q-value

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s, a'; \theta, \alpha) \right)$$

where:

- θ : is shared parameter for the network.
- α : parameterizes output stream for advantage function A .
- β : parameterizes output stream for value function V .

26.3.2 Model formulation

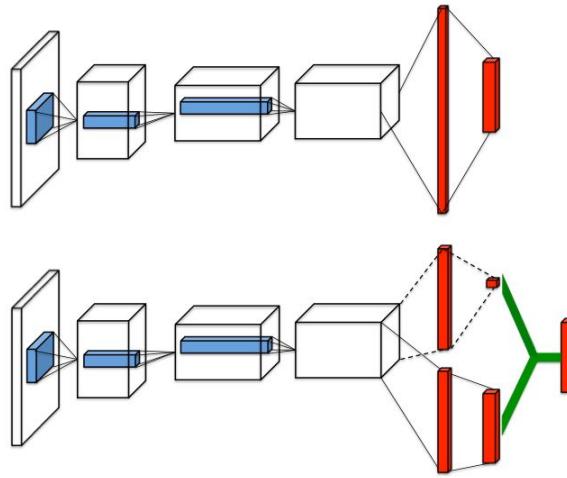


图 129: A popular single stream Q -network (top) and the dueling Q -network (bottom). The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action; the green output module implements above equation to combine them. Both networks output Q -values for each action.

Indeed, **dueling DQN define a model inference of Q -function**.

26.4 A Distributional Perspective on Reinforcement Learning (C51) (未完成)

26.4.1 Introduction

本节选自论文A Distributional Perspective on Reinforcement Learning[?]

26.4.2 Reference

- [mtomassoli.github.io:](https://mtomassoli.github.io/) Distributional RL

26.5 Prioritized Experience Replay

26.5.1 Introduction

本节选自论文Prioritized Experience Replay[?]

Background

Motivation

Contribution

Methodology

26.6 Distributed Prioritized Experience Replay

26.6.1 Introduction

本节选自论文Distributed Prioritized Experience Replay[?]

26.7 Rainbow: Combining Improvements in Deep Reinforcement Learning

26.7.1 Introduction

本节选自论文Rainbow: Combining Improvements in Deep Reinforcement Learning[?]

27 Model-Based Deep Reinforcement Learning

27.1 Benchmarking Model-Based Reinforcement Learning

27.1.1 Reference

- Benchmarking Model-Based Reinforcement Learning
- Paper: Benchmarking Model-Based Reinforcement Learning[?]

28 Multi-Agent Deep Reinforcement Learning

28.1 QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning

28.1.1 Introduction

本节选自QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning[?]

29 Imitation Learning

29.1 A Divergence Minimization Perspective on Imitation Learning Methods

29.1.1 Introduction

论文出处：[A Divergence Minimization Perspective on Imitation Learning Methods\[?\]](#)

总体来说，Imitation Learning 包括两部分：

1. **Behavioural Cloning (BC)**：从领域知识 (demonstrations) 中学习被视为一个监督学习问题 (supervised learning problem)，策略通过拟合专家领域知识的数据集 (dataset of expert demonstrations)，最终输出专家行为 (expert actions)。
2. **Inverse Reinforcement Learning (IRL)**：先推断出专家奖励函数 (reward function of the expert)，之后再训练策略函数以最大化这个专家奖励 (train a policy to optimize this reward)。

Understanding the Relation Among Imitation Learning Methods

1. Standard Behavioural Cloning
2. DAgger[?]
3. AIRL[?]
4. GAIL[?]

29.1.2 Model formulation

f-MAX: *f*-Divergence Max-Ent IRL

29.1.3 Reference

- Yue et al., 2018: Imitation Learning, ICML 2018 Tutorial

29.2 Imitation Learning from Imperfect Demonstration

29.2.1 Introduction

论文出处：[Imitation Learning from Imperfect Demonstration\[?\]](#)

30 Inverse Reinforcement Learning

30.1 Algorithms for Inverse Reinforcement Learning

30.1.1 Introduction

论文出处：[Algorithms for Inverse Reinforcement Learning\[?\]](#)

30.2 Imitation Learning from Imperfect Demonstration

30.2.1 Introduction

论文出处：[Imitation Learning from Imperfect Demonstration\[?\]](#)

Reinforcement Learning

$$\mathbb{E}_{\tau \sim \rho(\tau; \pi, \mathcal{P}, \mu)} \left[\sum_{t=0}^{\infty} \gamma^t r_t^\tau \right]$$

其中：

- $s_0^\tau \sim \mu(\cdot)$, $\forall \tau \in \mathcal{T}$
- $a_t^\tau \sim \pi(\cdot | s_t^\tau; \theta)$, $\forall \tau \in \mathcal{T}, \forall t \in \mathbb{N}_0$
- $s_{t+1}^\tau \sim \mathcal{P}(\cdot | s_t^\tau, a_t^\tau)$, $\forall \tau \in \mathcal{T}, \forall t \in \mathbb{N}_0$

Definition 30.1 (occupancy measure). 定义如下： $\rho_\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ 如下：

$$\rho_\pi(s, a) = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s | \pi)$$

其中 $\Pr(s_t = s | \pi)$ 是在策略 π 下，第 t 步的起始状态为 s 的概率密度（可理解 $\Pr(s_t | \pi)$ 为在 \mathcal{S} 空间内的概率分布）。 $\rho_\pi(s, a)$ 可理解为未归一化的 state-action pair 的密度 ($\mathcal{S} \times \mathcal{A}$ 空间内 logit 分布的一个维度)。

Theorem 8 (π 和 ρ_π 的关系). occupancy measure 在 imitation learning 中占有重要的地位，因为与策略 $\pi(a|s)$ 一一对应：

$$\pi(a|s) \triangleq \frac{\rho_\pi(s, a)}{\sum_{a' \in \mathcal{A}} \rho_\pi(s, a')}$$

Theorem 9 (normalized occupancy measure). 表示如下：

$$\begin{aligned} \rho_\pi(s, a) &\triangleq \frac{\rho_\pi(s, a)}{\sum_{s' \in \mathcal{S}, a' \in \mathcal{A}} \rho_\pi(s', a')} \\ &= \frac{\rho_\pi(s, a)}{\sum_{s' \in \mathcal{S}, a' \in \mathcal{A}} \left(\pi(a'|s') \sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s' | \pi) \right)} \\ &= \frac{\rho_\pi(s, a)}{\sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} \sum_{t=0}^{\infty} \pi(a'|s') \gamma^t \Pr(s_t = s' | \pi)} \\ &= \frac{\rho_\pi(s, a)}{\sum_{t=0}^{\infty} \gamma^t \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} \pi(a'|s_t = s') \Pr(s_t = s' | \pi)} \\ &= \frac{\rho_\pi(s, a)}{\sum_{t=0}^{\infty} \gamma^t \underbrace{\sum_{s' \in \mathcal{S}} \Pr(s_t = s' | \pi)}_1 \underbrace{\sum_{a' \in \mathcal{A}} \pi(a'|s_t = s')}_1} \\ &= \frac{\rho_\pi(s, a)}{\sum_{t=0}^{\infty} \gamma^t} \\ &= (1 - \gamma) \rho_\pi(s, a) \end{aligned}$$

Generative Adversarial Imitation Learning (GAIL) Imitation Learning 的问题设置是，给定专家领域经验样本集 (expert demonstration) $\mathcal{D} = \{(s_i, a_i)\}_{i=1}^N$

30.3 Discriminator-Actor-Critic: Addressing Sample Inefficiency and Reward Bias in Adversarial Imitation Learning (未完成)

30.3.1 Introduction

论文出处：[Discriminator-Actor-Critic: Addressing Sample Inefficiency and Reward Bias in Adversarial Imitation Learning\[?\]](#)

31 Safe Reinforcement Learning

31.1 Constrained Policy Optimization

31.1.1 Introduction

论文出处：[Constrained Policy Optimization\[?\]](#).

31.2 Safe Exploration in Continuous Action Spaces

31.2.1 Introduction

论文出处：[Safe Exploration in Continuous Action Spaces\[?\]](#)

31.3 Constrained Cross-Entropy Method for Safe Reinforcement Learning

31.3.1 Introduction

论文出处：[Constrained Cross-Entropy Method for Safe Reinforcement Learning\[?\]](#)

32 Meta Reinforcement Learning

32.1 Meta Reinforcement Learning Overview

32.1.1 Introduction

论文出处：[Lil'Log: Meta Reinforcement Learning\[?\]](#)

Part V**Automated Machine Learning (AutoML)
Approach and Theory**

33 AutoML for Feature Selection

33.1 AutoCross: Automatic Feature Crossing for Tabular Data in Real-World Applications (未完成)

33.1.1 Introduction

论文出处：[AutoCross: Automatic Feature Crossing for Tabular Data in Real-World Applications\[?\]](#)，主要用于自动特征交叉 (automatic feature crossing)

Problem

34 AutoML for Model Architecture Selection

35 AutoML for Loss Function

35.1 AM-LFS: AutoML for Loss Function Search (未完成)

35.1.1 Introduction

论文出处：[AM-LFS: AutoML for Loss Function Search\[?\]](#)

36 AutoML for Optimizer Selection

37 AutoML for Hyper-Parameter Selection

Part VI

Knowledge Graph Model and Theory

38 Introduction of Knowledge Graph (未完成)

38.1 title

Part VII

Model Applications of Computer Vision

39 Semantic Segmentation (Image Segmentation)

39.1 Fully Convolutional Networks for Semantic Segmentation

39.1.1 Introduction

论文出处：[Fully Convolutional Networks for Semantic Segmentation\[?\]](#)。语义分割（图像分割）任务其实是对图片样本中的每一个像素点进行分类。

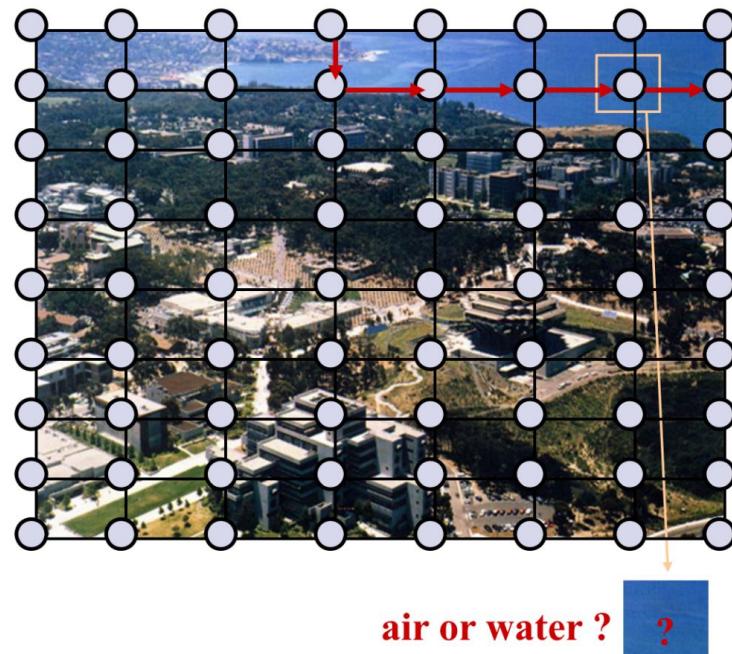


图 130: Using an Undirected Graphical Models (UGM) to predict whether a section of a scene is air or water.
Cite image from [Eric Xing-CMU 10-708, Spring 2014: Probabilistic Graphical Models](#)

39.1.2 Reference

- Jeremy Jordan: An overview of semantic image segmentation.

40 Image Synthesis

40.1 BigGAN (未完成)

40.1.1 Introduction

论文出处：[Large Scale GAN Training for High Fidelity Natural Image Synthesis\[?\]](#)

40.2 StyleGAN (未完成)

40.2.1 Introduction

本节选自论文A Style-Based Generator Architecture for Generative Adversarial Networks[?]

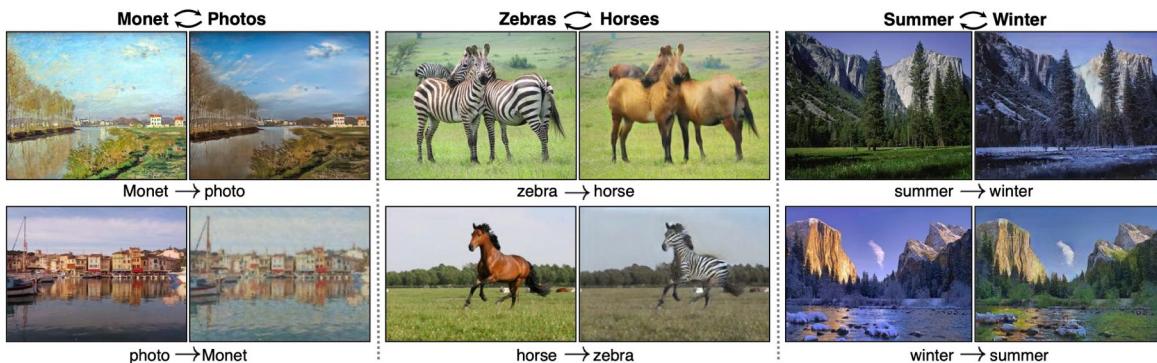
41 Image-to-Image Translation

41.1 CycleGAN

41.1.1 Introduction

论文出处：[Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks\[?\]](#)，用于解决[Image-to-Image Translation](#)的问题。而同一时期还有两篇姊妹文章[DualGAN\[?\]](#) 和[DiscoGAN\[?\]](#)，将在之后的章节中进行介绍。

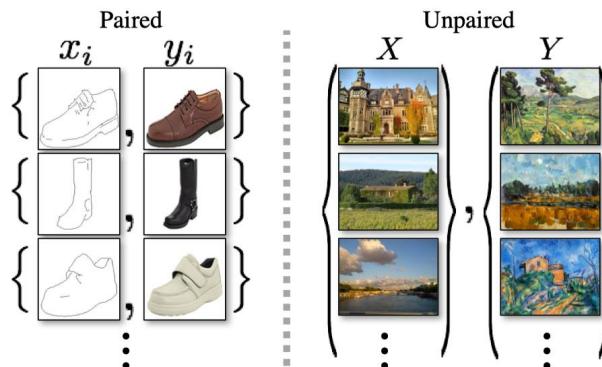
Paired training data is expansive and difficult to obtain 对于图像翻译 (image-to-image translation)、样式迁移 (style transfer) 类的实际问题，利用深度学习进行建模时，自然想到的方法是利用大量成对的训练样本 (paired training data) 进行训练，从而使得模型可以拟合出两种 domain 之间的映射关系。成对的训练样本如图所示：



但在现实中这种想法很不现实，因为其获取成本太高了：

- 除样式外完全一致场景的成对样本几乎没有
- 如果要得到这样的样本，花费的人力、物力、财力以及时间都是极其庞大的，完全不现实

因此，由于受到现实成对样本缺失的限制，我们不可能从成对的样本角度进行模型学习，所以，得想办法利用非成对样本 (unpaired training data) 进行学习，因为现实中，非成对的样本太多了，几乎所有情况都有。



Contribution 因此本文提出了一种在非成对样本集 $\{\mathbf{x}^{(i)}\}_{i=1}^N, \mathbf{x}^{(i)} \in \mathcal{X}$ 和 $\{\mathbf{y}^{(j)}\}_{j=1}^M, \mathbf{y}^{(j)} \in \mathcal{Y}$ 上的学习模型，模型在 \mathcal{Y} 空间的训练样本中抽取其独有的特征，并指出这些特征如何转换到另一空间 \mathcal{X} 的样本集合上。

41.1.2 Model formulation

模型的目标是：给定两组不成对训练样本集 $\{x^{(i)}\}_{i=1}^N, x^{(i)} \in \mathcal{X}$ 和 $\{y^{(j)}\}_{j=1}^M, y^{(j)} \in \mathcal{Y}$ 的前提下，学习两个 domains 之间的映射函数。其中：

- 两组特征空间维度 \mathcal{X} 和 \mathcal{Y} 不必相同
- 两组训练样本个数 N 和 M 不必相同

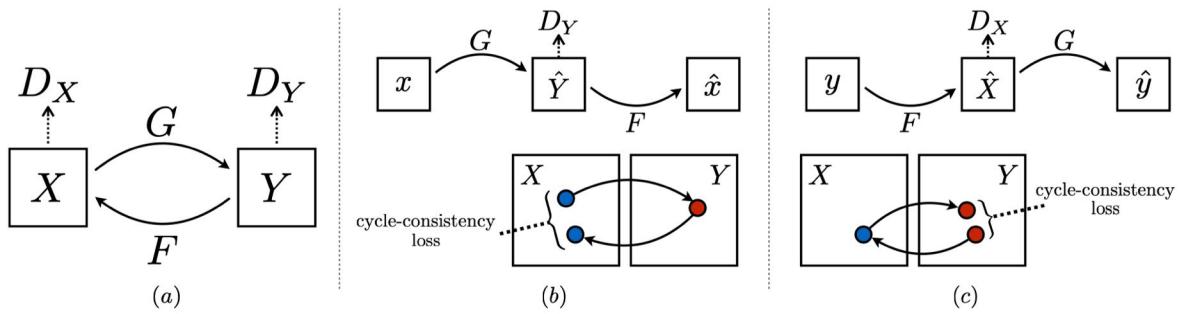


图 131: (a) CycleGAN model contains two mapping functions $G : \mathcal{X} \rightarrow \mathcal{Y}$ and $F : \mathcal{Y} \rightarrow \mathcal{X}$, and associated adversarial discriminators D_Y and D_X . D_Y encourages G to translate \mathcal{X} into outputs indistinguishable from domain \mathcal{Y} , and vice versa for D_X and F . To further regularize the mappings, we introduce two cycle consistency losses that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $\mathbf{x} \rightarrow G(\mathbf{x}) \rightarrow F(G(\mathbf{x})) \approx \mathbf{x}$, and (c) backward cycle-consistency loss: $\mathbf{y} \rightarrow G(\mathbf{y}) \rightarrow F(G(\mathbf{y})) \approx \mathbf{y}$

Adversarial Loss 基本的模型结构包含两个 GAN 网络：

1. 针对 $\mathcal{X} \rightarrow \mathcal{Y}$ 的 GAN:

- real inputs : $\{y^{(j)}\}_{j=1}^M$
- noise inputs : $\{x^{(i)}\}_{i=1}^N$ (对应传统 GAN 中的 z)
- generator: G
- discriminator: D_Y

2. 针对 $\mathcal{Y} \rightarrow \mathcal{X}$ 的 GAN:

- real inputs : $\{x^{(i)}\}_{i=1}^N$
- noise inputs : $\{y^{(j)}\}_{j=1}^M$ (对应传统 GAN 中的 z)
- generator: F
- discriminator: D_X

因此 $\mathcal{X} \rightarrow \mathcal{Y}$ 的 GAN，目标函数为：

$$\mathcal{L}_{\text{GAN}}(G, D_Y, \mathcal{X}, \mathcal{Y}) = \mathbb{E}_{y \sim p_{\text{data}}(y)} [\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log (1 - D_Y(G(x)))]$$

同理， $\mathcal{Y} \rightarrow \mathcal{X}$ 的 GAN，目标函数为：

$$\mathcal{L}_{\text{GAN}}(F, D_{\mathcal{X}}, \mathcal{Y}, \mathcal{X}) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D_{\mathcal{X}}(\mathbf{x})] + \mathbb{E}_{\mathbf{y} \sim p_{\text{data}}(\mathbf{y})} [\log (1 - D_{\mathcal{X}}(F(\mathbf{y})))]$$

因此，按理说模型的目标函数应该为：

$$G^*, F^* = \arg \min_{G, F} \max_{D_{\mathcal{Y}}, D_{\mathcal{X}}} \mathcal{L}(G, F, D_{\mathcal{X}}, D_{\mathcal{Y}}) = \mathcal{L}_{\text{GAN}}(G, D_{\mathcal{Y}}, \mathcal{X}, \mathcal{Y}) + \mathcal{L}_{\text{GAN}}(F, D_{\mathcal{X}}, \mathcal{Y}, \mathcal{X})$$

Cycle Consistency Loss 按理说，上述目标函数可以很好地学到样本空间之间的映射函数 $G : \mathcal{X} \rightarrow \mathcal{Y}$ 以及 $F : \mathcal{Y} \rightarrow \mathcal{X}$ ，但事实并非如此，在模型容量 (capacity) 足够大的情况下，网络可以将相同样本集合 **随机多方式（训练样本不同，随机种子不同 \Rightarrow 映射函数不同）** 地映射到目标样本空间中，并使得输出分布与目标分布相同。因此通过上述目标函数训练得到的 G 和 F 是极其**不稳定**的，其几乎铁定不能保证：对于训练样本外的特定的输入 \mathbf{x} ，能通过 G 得到期望的 \mathbf{y} ， F 也是一样。因此当务之急是减少映射函数的空间 (reduce the space of possible mapping functions)，所以引入了 Cycle Consistency，其包括了两部分：

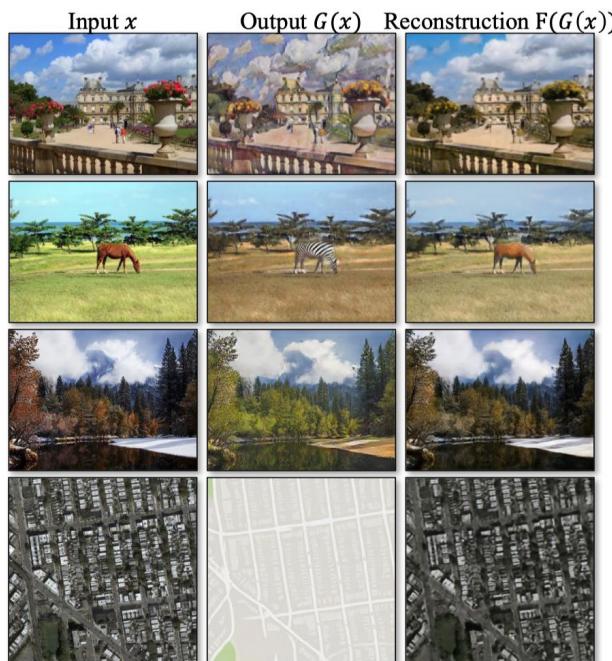
- forward cycle consistency: $\mathbf{x} \rightarrow G(\mathbf{x}) \rightarrow F(G(\mathbf{x})) \approx \mathbf{x}$
- backward cycle consistency: $\mathbf{y} \rightarrow G(\mathbf{y}) \rightarrow F(G(\mathbf{y})) \approx \mathbf{y}$

因此设置目标函数时，针对生成器 G 和 F 的训练，设置如下 cycle consistency loss：

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\|F(G(\mathbf{x})) - \mathbf{x}\|_1] + \mathbb{E}_{\mathbf{y} \sim p_{\text{data}}(\mathbf{y})} [\|G(F(\mathbf{y})) - \mathbf{y}\|_1]$$

注意

- 上述 Cycle Consistency 的形象解释：世界 (capacity) 那么大，随机地找，从北京 \mathcal{X} 出发，条条大路通罗马 \mathcal{Y} ，但是不一定能找到原路从罗马 \mathcal{Y} 回到北京 \mathcal{X} ，因此需要设置选择路径的条件，保证从哪出发，到终点后还能回到哪儿。



- 论文中作者使用了 L1 norm 是因为设置别的 cycle consistency loss 并没有取得更好的效果。事实上一系列 GAN 的 cycle consistency 距离定义均采用 L1 norm
- cycle consistency loss 的设置，还可以选择：
 - $F(G(\mathbf{x}))$ 与 \mathbf{x} 之间， $G(F(\mathbf{y}))$ 与 \mathbf{y} 之间的 l2-norm
 - $F(G(\mathbf{x}))$ 与 \mathbf{x} 之间， $G(F(\mathbf{y}))$ 与 \mathbf{y} 之间的 adversarial loss

Full Objective 因此涵盖了 Adversarial Loss 和 Cycle Consistency Loss，模型的最终目标函数为：

$$\mathcal{L}(G, F, D_{\mathcal{X}}, D_{\mathcal{Y}}) = \mathcal{L}_{\text{GAN}}(G, D_{\mathcal{Y}}, \mathcal{X}, \mathcal{Y}) + \mathcal{L}_{\text{GAN}}(F, D_{\mathcal{X}}, \mathcal{Y}, \mathcal{X}) + \lambda \mathcal{L}_{\text{cyc}}(G, F)$$

则最终得到的最优生成器 G^* 和 F^* 为：

$$G^*, F^* = \arg \min_{G, F} \max_{D_{\mathcal{Y}}, D_{\mathcal{X}}} \mathcal{L}(G, F, D_{\mathcal{X}}, D_{\mathcal{Y}})$$

Training details

- 目标函数中 $\mathcal{L}_{\text{GAN}}(G, D_{\mathcal{Y}}, \mathcal{X}, \mathcal{Y})$ 和 $\mathcal{L}_{\text{GAN}}(F, D_{\mathcal{X}}, \mathcal{Y}, \mathcal{X})$ 采用更加稳定的 least-squares loss，用于取代 negative log likelihood objective，那么此时 G 和 D 的目标函数需要分开设置即：
 - 训练 D ：

$$\min \mathbb{E}_{\mathbf{y} \sim p_{\text{data}}(\mathbf{y})} [(D(\mathbf{y}) - 1)^2] + \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [D(G(\mathbf{x}))^2]$$
 注意对于一个样本 \mathbf{y} ，判别器 D 函数输出的是标量 logit 值。
 - 训练 G ：

$$\min \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [(D(G(\mathbf{x})) - 1)^2]$$
- 为了保证 Cycle Consistency，需要将目标函数中的惩罚系数 λ 设置大一些，论文中设置 $\lambda = 10$
- Adam 优化算法，学习率设置为 0.0002，batch_size 设置为 1，每 100 个 epochs 学习率进行线性衰减

Experimental results and evaluation 实验展示了训练结束后测试阶段生成器的对样本的泛化性能

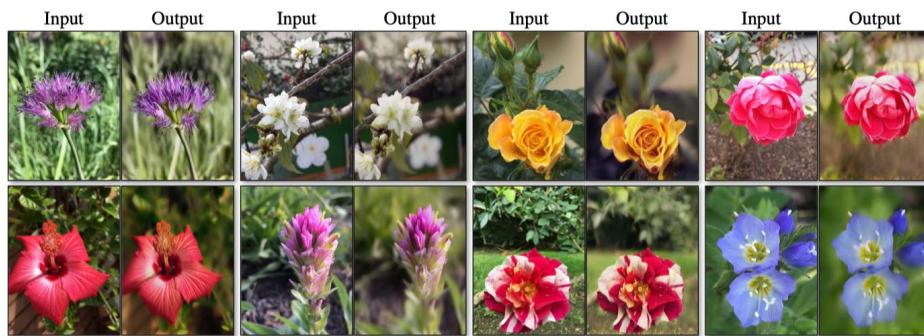


图 132: Photo enhancement: mapping from a set of smartphone snaps to professional DSLR photographs, the system often learns to produce shallow focus. Here we show some of the most successful results in our test set—average performance is considerably worse. See [website](#) for more comprehensive and random examples.

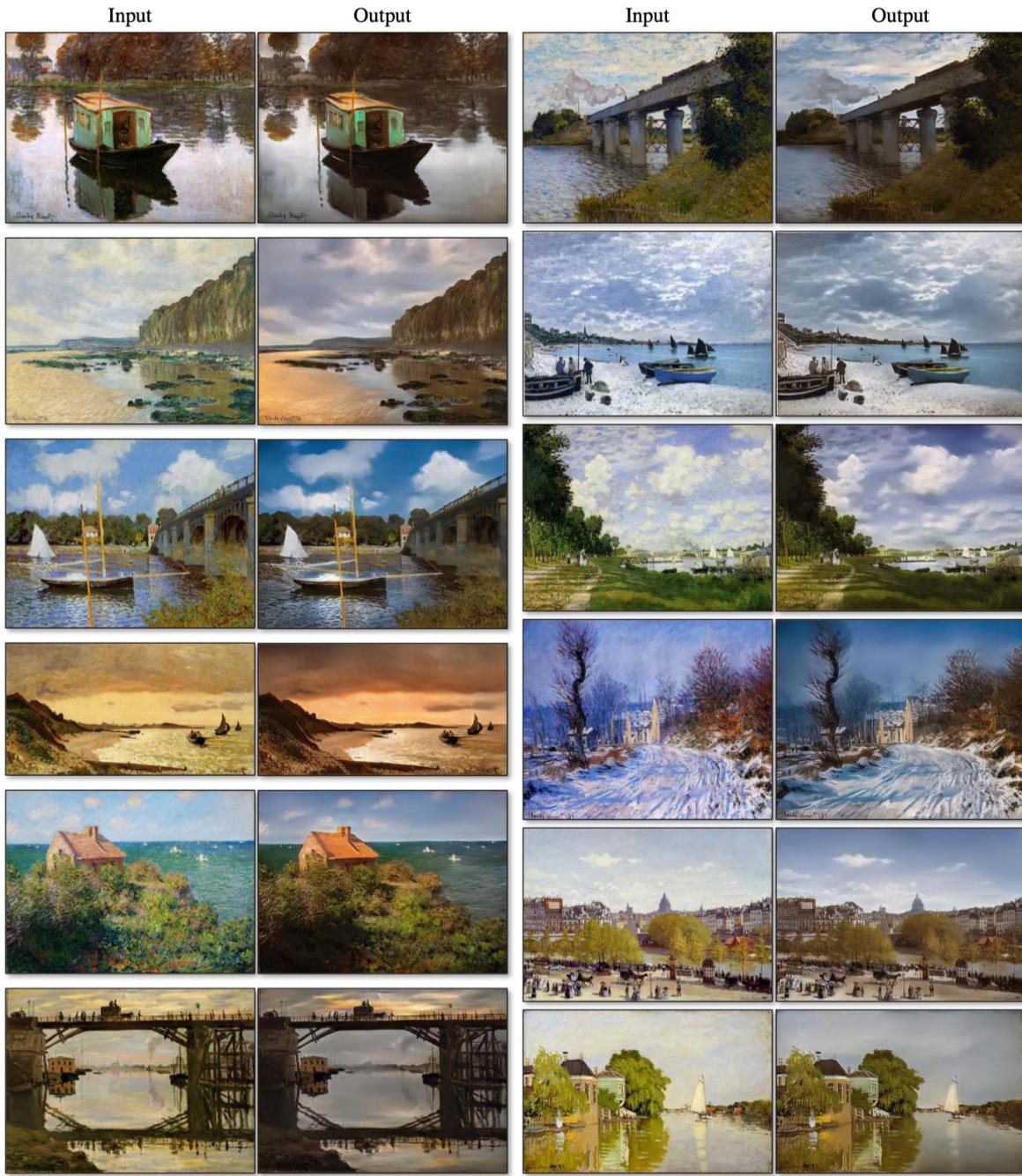


图 133: Relatively successful results on mapping Monet ' s paintings to a photographic style. Please see [website](#) for additional examples.

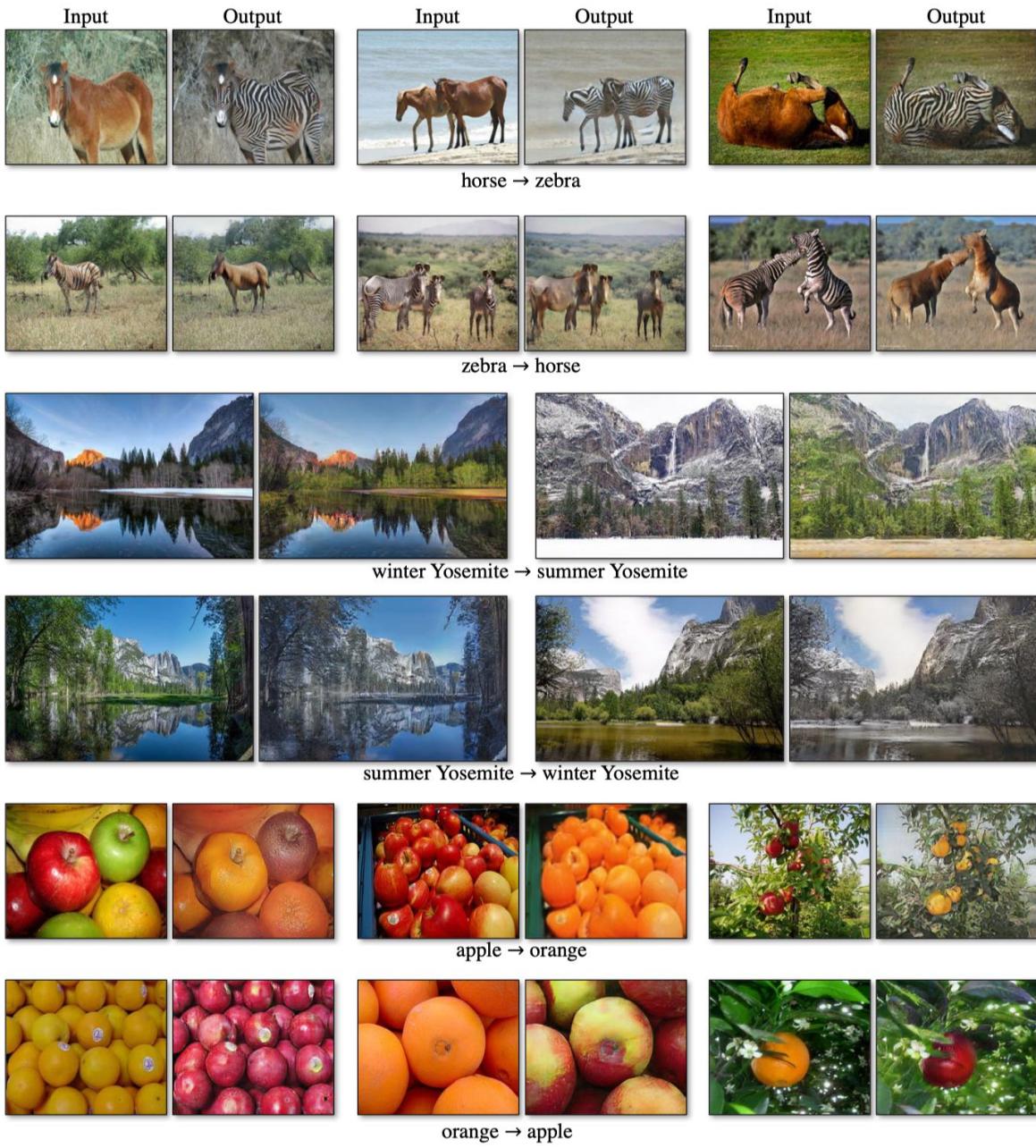


图 134: The method applied to several translation problems. These images are selected as relatively successful results please see [website](#) for more comprehensive and random results. In the top two rows, we show results on object transfiguration between horses and zebras, trained on 939 images from the wild horse class and 1177 images from the zebra class in Imagenet[?]. Also check out the horse→zebra demo [video](#). The middle two rows show results on season transfer, trained on winter and summer photos of Yosemite from Flickr. In the bottom two rows, we train our method on 996 apple images and 1020 navel orange images from ImageNet.

41.1.3 Reference

- Understanding and Implementing CycleGAN in TensorFlow
- Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

41.2 Augmented CycleGAN (未完成)

41.2.1 Introduction

本文选自 Augmented CycleGAN: Learning Many-to-Many Mappings from Unpaired Data[?]

41.3 DualGAN

41.3.1 Introduction

本文选自 DualGAN: Unsupervised Dual Learning for Image-to-Image Translation[?]，用于解决 **Image-to-Image Translation** 的问题。与上一节的 CycleGAN 类似，本文也提出了一种基于非成对无标签 (unpaired and unlabeled) 数据集的无监督图像翻译通用框架 (unsupervised learning framework for general-purpose image-to-image translation)。

41.3.2 Model formulation

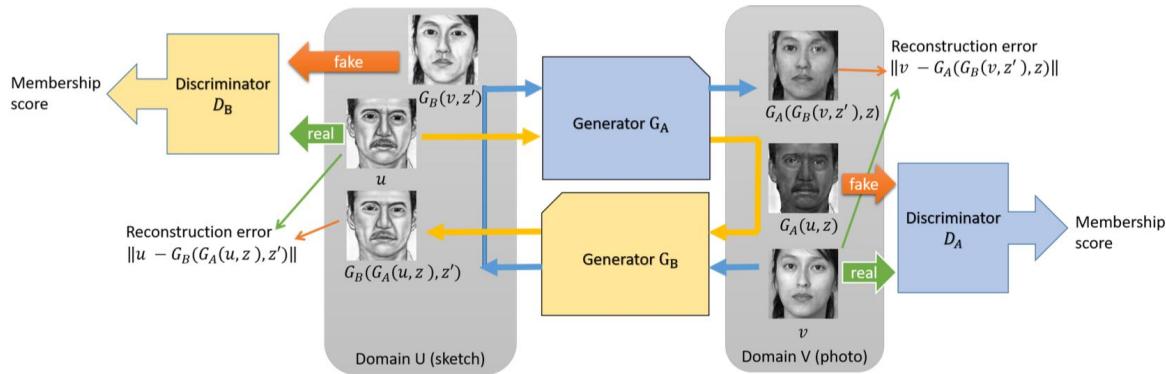


图 135: Network architecture and data flow chart of DualGAN for image-to-image translation.

Methodology 给定两组来自于 domain: \mathcal{U} 和 \mathcal{V} 的数据集，分别记作： $\{\mathbf{u}^{(i)}\}_{i=1}^N$ 和 $\{\mathbf{v}^{(j)}\}_{j=1}^m$ ，DualGAN 实际上包含了两个 GANs 的学习任务，分别记作 primal task 和 dual task。

- 对于 primal task：学习一个生成器 $G_A : \mathcal{U} \rightarrow \mathcal{V}$ (主要目的) 和一个判别器 $D_A : \mathcal{V} \rightarrow \mathbb{R}$ (次要目的)
- 对于 dual task：学习一个生成器 $G_B : \mathcal{V} \rightarrow \mathcal{U}$ (主要目的) 和一个判别器 $D_B : \mathcal{U} \rightarrow \mathbb{R}$ (次要目的)

注意 这里判别器映射为 $D_A : \mathcal{V} \rightarrow \mathbb{R}$ 和 $D_B : \mathcal{U} \rightarrow \mathbb{R}$ ，而不是 $D_A : \mathcal{V} \rightarrow \{0, 1\}$ 和 $D_B : \mathcal{U} \rightarrow \{0, 1\}$ ，是参考了 WGAN 对判别器目标函数的改进方式，即输出的是 logit 函数值，之所以不选择 sigmoid cross-entropy loss 是因为 sigmoid 函数会带来局部饱和 (locally saturated) 的问题，从而引发梯度消失 (vanishing gradients)。两个 tasks 具体的学习方式如下：

- primal task：对于任意一个特定的样本 $\mathbf{u} \in \mathcal{U}$ ：
 - 利用加噪生成器 $G_A : \mathcal{U} \rightarrow \mathcal{V}$ 作用于样本 $\mathbf{u} \in \mathcal{U}$ ，产生生成样本 $G_A(\mathbf{u}, \mathbf{z}) \in \mathcal{V}$ ，其中 $\mathbf{z} \in \mathcal{U}$ 为随机噪声
 - 利用 $D_A : \mathcal{V} \rightarrow \mathbb{R}$ 对生成样本 $G_A(\mathbf{u}, \mathbf{z}) \in \mathcal{V}$ 进行评价
 - 利用加噪生成器 $G_B : \mathcal{V} \rightarrow \mathcal{U}$ 对生成样本 $G_A(\mathbf{u}, \mathbf{z}) \in \mathcal{V}$ 进行复原，得到 $G_B(G_A(\mathbf{u}, \mathbf{z}), \mathbf{z}') \in \mathcal{U}$ 作为原始样本 $\mathbf{u} \in \mathcal{U}$ 的重构
 - 利用 reconstruction loss : $\|G_B(G_A(\mathbf{u}, \mathbf{z}), \mathbf{z}') - \mathbf{u}\|$ 对重构效果进行衡量
- dual task：对于任意一个特定的样本 $\mathbf{v} \in \mathcal{V}$ ：

1. 利用加噪生成器 $G_B : \mathcal{V} \rightarrow \mathcal{U}$ 作用于样本 $\mathbf{v} \in \mathcal{V}$ ，产生生成样本 $G_B(\mathbf{v}, \mathbf{z}') \in \mathcal{U}$ ，其中 $\mathbf{z}' \in \mathcal{V}$ 为随机噪声
2. 利用 $D_B : \mathcal{U} \rightarrow \mathbb{R}$ 对生成样本 $G_B(\mathbf{v}, \mathbf{z}') \in \mathcal{U}$ 进行评价
3. 利用加噪生成器 $G_A : \mathcal{U} \rightarrow \mathcal{V}$ 对生成样本 $G_B(\mathbf{v}, \mathbf{z}') \in \mathcal{U}$ 进行复原，得到 $G_A(G_B(\mathbf{v}, \mathbf{z}'), \mathbf{z}) \in \mathcal{V}$ 作为原始样本 $\mathbf{v} \in \mathcal{V}$ 的重构
4. 利用 reconstruction loss : $\|G_A(G_B(\mathbf{v}, \mathbf{z}'), \mathbf{z}) - \mathbf{v}\|$ 对重构效果进行衡量

Objective 针对随机配对的一对训练样本 $(\mathbf{u}, \mathbf{v}), \mathbf{u} \in \mathcal{U}, \mathbf{v} \in \mathcal{V}$:

- adversarial loss 为：

$$\min_{G_A} \max_{D_A} \mathcal{L}_{\text{GAN}}(G_A, D_A; \mathbf{u}, \mathbf{v}) = D_A(\mathbf{v}) - D_A(G_A(\mathbf{u}, \mathbf{z}))$$

$$\min_{G_B} \max_{D_B} \mathcal{L}_{\text{GAN}}(G_B, D_B; \mathbf{v}, \mathbf{u}) = D_B(\mathbf{u}) - D_B(G_B(\mathbf{v}, \mathbf{z}'))$$

- reconstruction loss 为：

$$\min_{G_A, G_B} \mathcal{L}_{\text{recon}}(G_A, G_B; \mathbf{u}, \mathbf{v}) = \lambda_{\mathcal{U}} \|\mathbf{u} - G_B(G_A(\mathbf{u}, \mathbf{z}), \mathbf{z}')\| + \lambda_{\mathcal{V}} \|\mathbf{v} - G_A(G_B(\mathbf{v}, \mathbf{z}'), \mathbf{z})\|$$

reconstruction loss 中衡量 recovery error 使用的是 L1 距离，因为 L2 距离会导致模糊 [?], 而无论是 DualGAN，还是 CycleGAN，对 recovery error 的定义都使用了 L1 距离。关于惩罚系数 $\lambda_{\mathcal{U}}$ 和 $\lambda_{\mathcal{V}}$:

- $\lambda_{\mathcal{U}}$ 和 $\lambda_{\mathcal{V}}$ 取值一般在 [100.0, 1000.0] 之间
- 如果 \mathcal{U} 包含天然照片而 \mathcal{V} 不包含，则设置 $\lambda_{\mathcal{U}}$ 小于 $\lambda_{\mathcal{V}}$

因此最终的目标函数为：

$$\min_{G_A, G_B} \max_{D_A, D_B} \mathcal{L}(G_A, G_B, D_A, D_B; \mathbf{u}, \mathbf{v}) = \mathcal{L}_{\text{GAN}}(G_A, D_A; \mathbf{u}, \mathbf{v}) + \mathcal{L}_{\text{GAN}}(G_B, D_B; \mathbf{v}, \mathbf{u}) + \mathcal{L}_{\text{recon}}(G_A, G_B; \mathbf{u}, \mathbf{v})$$

Training procedure 训练沿袭了 WGAN 的训练方法，对于一些基本超参数的选择说明如下：

- 使用 RMSProp 优化器进行参数的优化，而不是使用 Adam 这样基于动量的算法
- n_{critic} 设置为 2 – 4
- batch size 设置为 1 – 4 (GAN 相关问题的 batch size 都设置得很小)
- clipping parameter c 一般设置在 [0.01, 0.1] 之间，具体的根据具体问题的不同而调整

Algorithm 43: DualGAN training procedure[?]

Input: Image set U ;
 Image set V ;
 GAN A with generator parameters θ_A and discriminator parameters ω_A ;
 GAN B with generator parameters θ_B and discriminator parameters ω_B ;
 clipping parameter c ;
 the number of iterations of the critic per generator iteration n_{critic} ;
 the batch size m .

Output: The final parameters of generator A: θ_A and generator B: θ_B

Randomly initialize $\theta_i, \omega_i, i \in \{A, B\}$.

repeat

for $t = 1, \dots, n_{\text{critic}}$ **do**

sample images $\{\mathbf{u}^{(k)}\}_{k=1}^m \subseteq U, \{\mathbf{v}^{(k)}\}_{k=1}^m \subseteq V$.
 update ω_A to maximize $\frac{1}{m} \sum_{k=1}^m \mathcal{L}_{\text{GAN}}(G_A, D_A; \mathbf{u}^{(k)}, \mathbf{v}^{(k)})$.
 update ω_B to maximize $\frac{1}{m} \sum_{k=1}^m \mathcal{L}_{\text{GAN}}(G_B, D_B; \mathbf{v}^{(k)}, \mathbf{u}^{(k)})$.
 clip parameters of both discriminator: $\text{clip}(\omega_A, -c, c), \text{clip}(\omega_B, -c, c)$.

end

sample images $\{\mathbf{u}^{(k)}\}_{k=1}^m \subseteq U, \{\mathbf{v}^{(k)}\}_{k=1}^m \subseteq V$.
 update θ_A, θ_B to minimize $\frac{1}{m} \sum_{k=1}^m \mathcal{L}(G_A, G_B, D_A, D_B; \mathbf{u}^{(k)}, \mathbf{v}^{(k)})$

until convergence;

Experimental results and evaluation 实验展示的为训练完成后的模型在测试阶段的效果，其中实验中的 Input 均为原始输入（即原始 GAN 中的 z ），DualGAN, GAN 以及 cGAN[?] 展示的都是生成器将原始输入映射后的结果，具体效果如下：



图 136: Experimental results for translating Chinese paintings to oil paintings (without GT available). The background grids shown in the GAN results imply that the outputs of GAN are not as stable as those of DualGAN.

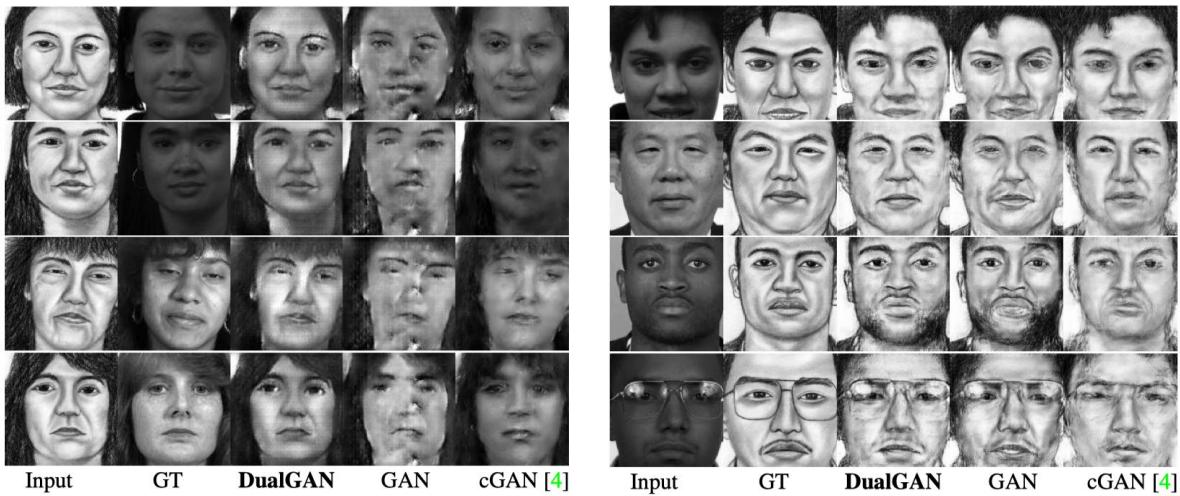


图 137: Photo→sketch translation for faces. Results of DualGAN are generally sharper than those from faces. More artifacts and blurriness are showing up cGAN, even though the former was trained using unpaired data, whereas the latter makes use of image correspondence.

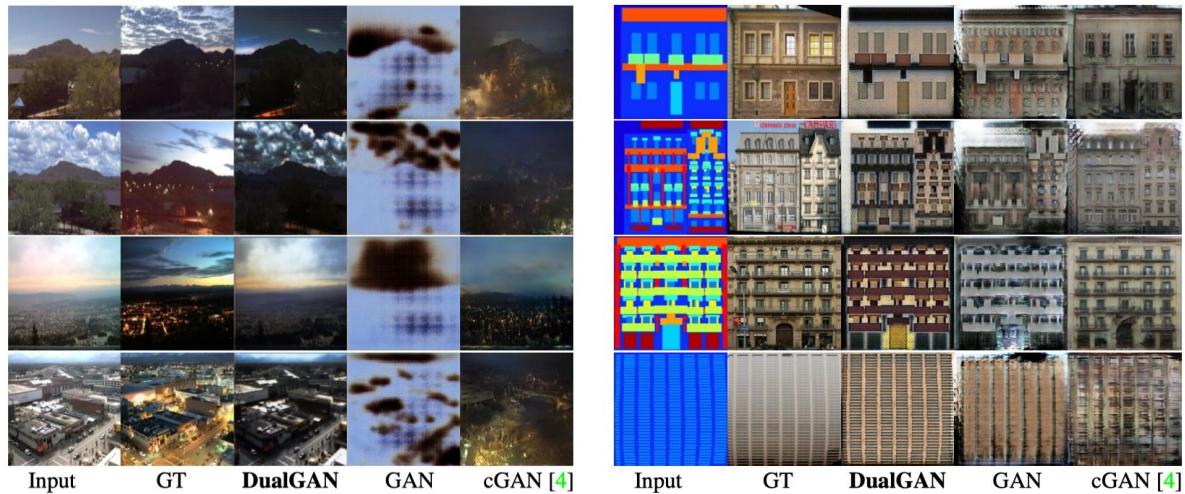


图 138: Results for sketch→photo translation of DualGAN are generally sharper than those from faces. More artifacts and blurriness are showing up cGAN, even though the former was trained using unpaired data, whereas the latter makes use of image correspondence.

图 139: Results of day→night translation. cGAN is trained with labeled(paired) data, whereas DualGAN is trained in an unsupervised manner. DualGAN successfully emulates the night well with the corresponding photos in finer details while preserving textures in the inputs, e.g., In contrast, results from GAN and cGAN contain see differences over the cloud regions between our many artifacts. Over regions with labelphoto misresults and the ground truth (GT). In comparison, alignment, cGAN often yields blurry output (e.g., results of cGAN and GAN contain much less details. the roof in second row and the entrance in third row).

图 140: Results of label→facade translation. DualGAN faithfully preserves the structures in the labelphoto misresults and the ground truth (GT). In comparison, alignment, cGAN often yields blurry output (e.g., results of cGAN and GAN contain much less details. the roof in second row and the entrance in third row).

41.4 DiscoGAN

41.4.1 Introduction

本文选自 Learning to Discover Cross-Domain Relations with Generative Adversarial Networks[?]，用于解决 Image-to-Image Translation 的问题。

Problem 与上述 CycleGAN 和 DualGAN 一样，DiscoGAN 解决的问题依旧是两个 image domains 之间的 image-to-image translation 问题。对于两个 image domains 之间的图像翻译问题，自然想到的办法还是监督学习 (supervised learning) 的方法譬如 cGAN[?]，但是，譬如 cGAN 等监督学习方法存在很大的问题：

- 监督的天然成对数据非常稀少
- 为了获取监督数据的打标签任务是极其劳动密集型 (labor intensive) 的，实际成本太大
- 如果一个 domain 的图片在另一个 domain 中缺少对应样本，或者存在多个最佳候选对应样本，那么这时训练就变得非常棘手

因此需要在没有任何显式成对匹配数据 (explicitly paired data) 的情况下，挖掘出两个 visual domains 之间的关系。

Contribution 本文提出了一种利用 GANs 发掘跨域之间关系 (cross-domain relations) 的模型：DiscoGAN，在训练阶段，模型通过两组没有显式成对标签信息的图像集合进行学习，并且不需要任何预训练 (pre-training)。在预测阶段，模型通过输入一个 domain 内的一张图片，产生其在另一个 domain 内的对应图片。

41.4.2 Model formulation

Notation

- $G_{AB} : \mathbb{R}_A^{64 \times 64 \times 3} \rightarrow \mathbb{R}_B^{64 \times 64 \times 3}$ (具体任务空间可不同，下同)
- $G_{BA} : \mathbb{R}_B^{64 \times 64 \times 3} \rightarrow \mathbb{R}_A^{64 \times 64 \times 3}$
- $D_B : \mathbb{R}_B^{64 \times 64 \times 3} \rightarrow [0, 1]$
- $D_A : \mathbb{R}_A^{64 \times 64 \times 3} \rightarrow [0, 1]$

Objective 可以将损失函数视为三部分：Generators 损失函数，Discriminators 损失函数以及 Reconstruction 损失函数

$$\min_{G_A, G_B, D_A, D_B} \mathcal{L} = \mathcal{L}_G + \mathcal{L}_D + \mathcal{L}_{Reconst}$$

其中 Reconstruction loss 可以为任何的距离定义方式：

- L_1 loss (一般采用)
- L_2 loss
- Huber loss

Experimental results and evaluation



图 141: This GAN-based model trains with two independently collected sets of images and learns how to map two domains without any extra label. In this paper, we reduce this problem into generating a new image of one domain given an image from the other domain. (a) shows a high-level overview of the training procedure of our model with two independent sets (e.g. handbag images and shoe images). (b) and (c) show results of our method. Our method takes a handbag (or shoe) image as an input, and generates its corresponding shoe (or handbag) image. Again, it's worth noting that our method does not take any extra annotated supervision and can self-discover relations between domains.

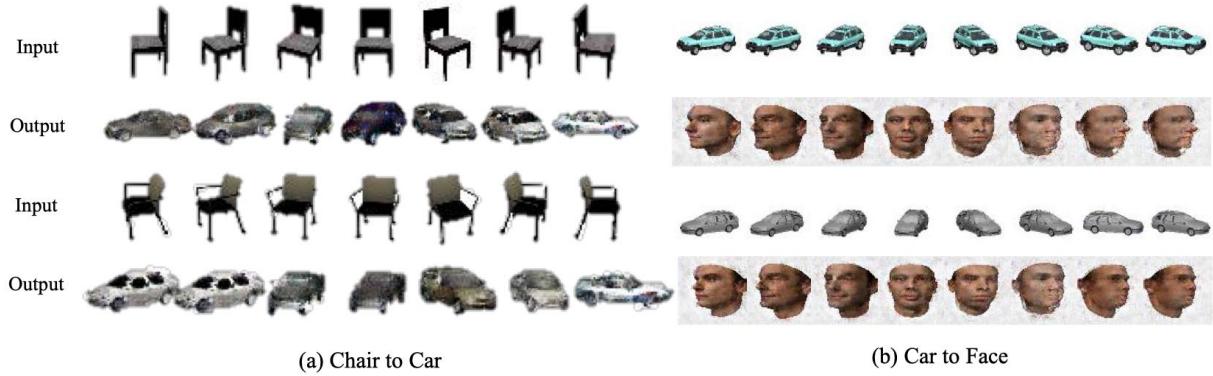


图 142: Discovering relations of images from visually very different object classes. (a) chair to car translation. DiscoGAN is trained on chair and car images (b) car to face translation. DiscoGAN is trained on car and face images. The model successfully pairs images with similar orientation.



图 143: (a,b) Translation of gender in Facescrub dataset and CelebA dataset. (c) Blond to black and black to blond hair color conversion in CelebA dataset. (d) Wearing eyeglasses conversion in CelebA dataset (e) Results of applying a sequence of conversion of gender and hair color (left to right) (f) Results of repeatedly applying the same conversions (upper: hair color, lower: gender)

42 Image Inpainting

42.1 Semantic Image Inpainting with Deep Generative Models (未完成)

42.1.1 Introduction

本文选自 Semantic Image Inpainting with Deep Generative Models[?]

43 Object Detection

Part VIII

Model Applications of Natural Language Processing

44 Word Representation

44.1 A Neural Probabilistic Language Model

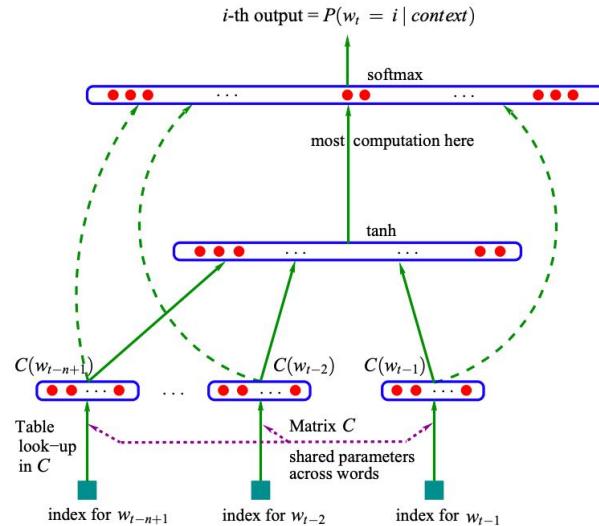
44.1.1 Introduction

本文选自 A Neural Probabilistic Language Model[?]

n-gram 对于一个句子（词组序列）： $(w^{(1)}, w^{(2)}, \dots, w^{(m)})$ ，其发生的概率为：

$$\begin{aligned} P(w^{(1)}, w^{(2)}, \dots, w^{(m)}) &= \prod_{i=1}^m P(w^{(i)} | w^{(1)}, \dots, w^{(i-1)}) \\ &\approx \prod_{i=1}^m P(w^{(i)} | w^{(i-(n-1))}, \dots, w^{(i-1)}) \end{aligned}$$

44.1.2 Model formulation



- input layer : 上下文词的 indices
- embedding layer : 输出每个上下文词对应的 embedding vector
- hidden layer : 是所有上下文词 embedding 向量的 concat，然后使用 $tanh$ 激活函数进行非线性变换
- output layer : 最终输出层采用 softmax 函数进行激活，输出在所有词上的概率分布 :

$$P(w^{(i)} | w^{(i-(n-1))}, \dots, w^{(i-1)}) = \frac{\exp(z_{w^{(i)}})}{\sum_{j=1}^V \exp(z_{w^{(j)}})}$$

44.2 Word2vec

本节来自论文：Distributed Representations of Words and Phrases and their Compositionality[?]。模型实际上由两个模型组成：

- CBOW: 根据上下文预测中心词
 - Skip-gram : 根据中心词预测上下文

学习之后每一个存在于训练语料中的单词都有一个对应的向量表示，意味该单词在语义空间中的位置 而学

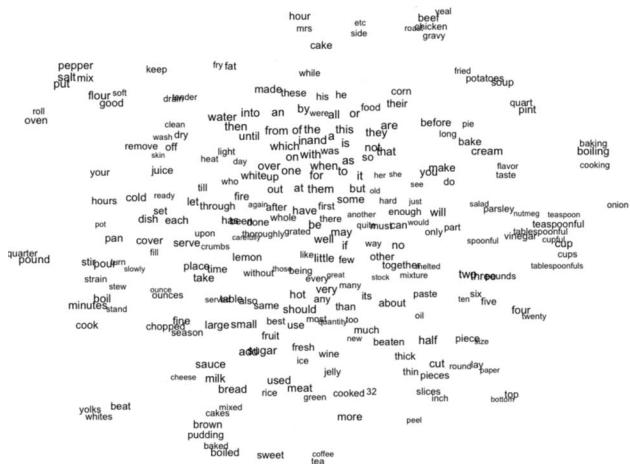


图 144: Word vector representation

习后我们希望相似单词在语义空间中有：

- 相近的绝对空间差（距离）
 - 类似的相对空间差（距离，方向）

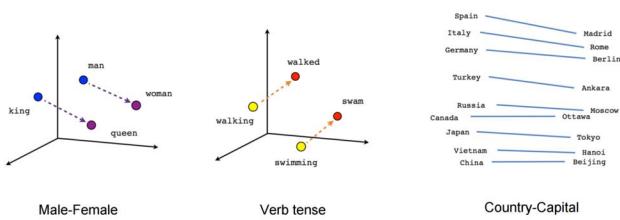


图 145: Example of word vectors

参考文章 [?] 内容，以下分别介绍这两个模型。

44.2.1 Continuous Bag-of-Word Model (CBOW)

One word context 由于 CBOW 是根据上下文 contexts 预测中心词 target，所以最简单的 One word context 版本的 CBOW 是根据一个上下文词 context 预测一个中心词 target (一对一)。

Dataset

$$\{(c^{(i)}, t^{(i)})\}_{i=1}^N$$

其中：

- $c^{(i)}$: 第 i 个训练样本的唯一上下文词 ids
- $t^{(i)}$: 第 i 个训练样本的唯一中心词 ids

譬如考虑句子：“今天你真美”，假设当前以你作为中心词 t ，设置滑窗大小为 2，则生成的训练数据集为：

$$\{(今, 你), (天, 你), (真, 你), (美, 你)\}$$

只不过实际计算机处理数据的时候将样本中一个个中文字符替换为了对应的唯一数字索引 ids。

Inference 有以下符号表示：

- V : 词表长度 (一共有多少个词)
- N : 隐层神经元个数 (词向量维度)
- $\mathbf{W} \in \mathbb{R}^{V \times N}$: 输入层和隐层之间全链接的权重。 \mathbf{W} 中的第 j 行 \mathbf{v}_j 即为第 j 个词的词向量
- $\mathbf{W}' \in \mathbb{R}^{N \times V}$: 隐层和输出层之间全链接的权重。

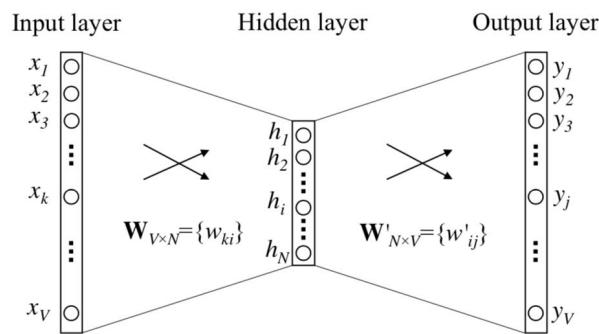


图 146: A simple CBOW model with only one word in the context

前向推导过程如下：

- 从输入层到隐层：

$$\mathbf{h} = \mathbf{x}\mathbf{W}^T = \mathbf{W}'_{(j,:)} := \mathbf{v}'_{\mathbf{w}_I}$$

解释：

- $\because \mathbf{W}^T \in \mathbb{R}^{\mathcal{N} \times \mathcal{V}}$, $\mathbf{x} \in \mathbb{R}^{\mathcal{V} \times \text{BatchSize}}$,
- $\therefore \mathbf{h} \in \mathbb{R}^{\mathcal{N} \times \text{BatchSize}}$
- 特别地, 如果 $\text{BatchSize} = 1 \Rightarrow \mathbf{h} \in \mathbb{R}^{\mathcal{N}}$ 。实质 \mathbf{h} 就是 one-hot 的输入向量 \mathbf{x} 中非零列索引 j 所对应的输入词 w_I 的词向量表示 $\mathbf{v}_{w_I}^T$
- **注意:** 激活函数为线性激活函数 (或者认为没有激活函数)

- 从隐层到输出层 :

$$\mathbf{u} = \mathbf{W}' \mathbf{h}$$

解释 :

- $\because \mathbf{W}' \in \mathbb{R}^{\mathcal{V} \times \mathcal{N}}$, $\mathbf{h} \in \mathbb{R}^{\mathcal{N} \times \text{BatchSize}}$
- $\therefore \mathbf{u} \in \mathbb{R}^{\mathcal{V} \times \text{BatchSize}}$
- 特别地, 如果 $\text{BatchSize} = 1 \Rightarrow \mathbf{u} \in \mathbb{R}^{\mathcal{V}}$ 。那么有 :

$$u_k = \mathbf{v}_{w_k}^T \mathbf{h}$$

其中 :

- * \mathbf{v}_{w_k}' 为 \mathbf{W}' 的第 k 列
- * $\mathbf{v}_{w_k}'^T$ 为 \mathbf{W}'^T 的第 k 行

- 从输出层到概率输出层 :

$$p(w_k | w_I) = \hat{y}_k = \frac{\exp(u_k)}{\sum_{k'=1}^{\mathcal{V}} \exp(u_{k'})}$$

解释 :

- $\because u_k = \mathbf{v}_{w_k}^T \mathbf{h}$, $\mathbf{h} := \mathbf{v}_{w_I}$
- $\therefore u_k = \mathbf{v}_{w_k}^T \mathbf{v}_{w_I}$
- \mathbf{v}_w , $\mathbf{v}_{w'}$ 均为词 w 的向量表示, 其中输入向量 \mathbf{v}_w 来自于从输入层到隐层的参数 \mathbf{W} , 输出向量 $\mathbf{v}_{w'}$ 来自于从隐层到输出层的参数 \mathbf{W}'

因此有 :

$$\therefore p(w_k | w_I) = \frac{\exp(\mathbf{v}_{w_k}^T \mathbf{v}_{w_I})}{\sum_{k'=1}^{\mathcal{V}} \exp(\mathbf{v}_{w_k}^T \mathbf{v}_{w_I})}$$

Objective function 目标函数为最大化条件概率, (以单一训练样本进行说明) 即:

$$\begin{aligned} & \max p(w_O | w_I) \\ \Rightarrow & \max \log p(w_O | w_I) \\ \Rightarrow & \max \log \frac{\exp(\mathbf{v}_{w_O}^T \mathbf{v}_{w_I})}{\sum_{k=1}^{\mathcal{V}} \exp(\mathbf{v}_{w_k}^T \mathbf{v}_{w_I})} \end{aligned}$$

$$\begin{aligned}
&\Rightarrow \max \log(\exp(\mathbf{v}'^T_{w_O} \mathbf{v}_{w_I})) - \log \sum_{k=1}^V \exp(\mathbf{v}'^T_{w_k} \mathbf{v}_{w_I}) \\
&\Rightarrow \max \mathbf{v}'^T_{w_O} \mathbf{v}_{w_I} - \log \sum_{k=1}^V \exp(\mathbf{v}'^T_{w_k} \mathbf{v}_{w_I}) \\
&\Rightarrow \min J = \log \sum_{k=1}^V \exp(\mathbf{v}'^T_{w_k} \mathbf{v}_{w_I}) - \mathbf{v}'^T_{w_O} \mathbf{v}_{w_I}
\end{aligned}$$

Optimization 以神经网络的结构进行反向剖析，那么按照网络结构书写目标函数为交叉熵：

$$\min J = - \sum_{k=1}^V y_k \log \hat{y}_k \quad (\text{单一样本})$$

更新隐层 \Rightarrow 输出层的参数：

$$\begin{aligned}
\frac{\partial J}{\partial w'_{lk}} &= \frac{\partial J}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial u_k} \frac{\partial u_k}{\partial w'_{lk}} \\
\therefore \frac{\partial J}{\partial \hat{y}_k} &= \frac{\partial}{\partial \hat{y}_k} \left(- \sum_{k=1}^V y_k \log \hat{y}_k \right) \\
&= \frac{\partial}{\partial \hat{y}_k} \left(- y_k \log \hat{y}_k \right) = - \frac{y_k}{\hat{y}_k} \\
\therefore \frac{\partial \hat{y}_k}{\partial u_k} &= \frac{\partial}{\partial u_k} \left(\frac{e^{u_k}}{\sum_{k'=1}^V e^{u_{k'}}} \right) \\
&= \frac{\frac{\partial}{\partial u_k} (e^{u_k}) \sum_{k'=1}^V e^{u_{k'}} - e^{u_k} \frac{\partial}{\partial u_k} \left(\sum_{k'=1}^V e^{u_{k'}} \right)}{\left(\sum_{k'=1}^V e^{u_{k'}} \right)^2} \\
&= \frac{e^{u_k} \sum_{k'=1}^V e^{u_{k'}} - e^{u_k} e^{u_k}}{\left(\sum_{k'=1}^V e^{u_{k'}} \right)^2} \\
&= \frac{e^{u_k} \sum_{k'=1}^V e^{u_{k'}} - e^{u_k} e^{u_k}}{\sum_{k'=1}^V e^{u_{k'}} \sum_{k'=1}^V e^{u_{k'}}} = \hat{y}_k (1 - \hat{y}_k) \\
\therefore \frac{\partial u_k}{\partial w'_{lk}} &= \frac{\partial}{\partial w'_{lk}} \left(\sum_{l'=1}^N w'_{l'k} h_{l'} \right) = h_l \\
\therefore \frac{\partial J}{\partial w'_{lk}} &= - \frac{y_k}{\hat{y}_k} \hat{y}_k (1 - \hat{y}_k) h_l = - y_k (1 - \hat{y}_k) h_l
\end{aligned}$$

更新输入层 \Rightarrow 隐层的参数：

$$\begin{aligned}
\frac{\partial J}{\partial w_{jl}} &= \frac{\partial J}{\partial h_l} \frac{\partial h_l}{\partial w_{jl}} \\
&= \left(\sum_{k=1}^V \frac{\partial J}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial u_k} \frac{\partial u_k}{\partial h_l} \right) \frac{\partial h_l}{\partial w_{jl}}
\end{aligned}$$

$$\begin{aligned} \because \frac{\partial u_k}{\partial h_l} &= w'_{lk}, \quad \frac{\partial h_l}{\partial w_{jl}} = x_j \\ \therefore \frac{\partial J}{\partial w_{jl}} &= \left(\sum_{k=1}^V -y_k(1 - \hat{y}_k)w'_{lk} \right)x_j \\ &= -x_j \sum_{k=1}^V y_k(1 - \hat{y}_k)w'_{lk} \end{aligned}$$

Multi-word context

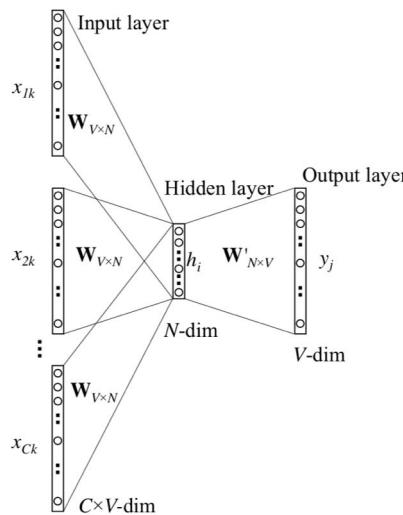


图 147: A CBOW model with multi words in the context

Inference

- 输入层 \Rightarrow 隐层：使用多个 embedding 的均值作为隐层 h ：

$$\begin{aligned} \mathbf{h} &= \frac{1}{C} \mathbf{W}^T (\mathbf{x}_1 + \mathbf{x}_2 + \dots + \mathbf{x}_C) \\ &= \frac{1}{C} (\mathbf{v}_{w_1} + \mathbf{v}_{w_2} + \dots + \mathbf{v}_{w_C})^T \end{aligned}$$

注意： 图中显示所有的上下文单次共享唯一一个输入层到隐层的参数矩阵 $\mathbf{W} \in \mathbb{R}^{V \times D}$ ，但是实际程序中没这么复杂， \mathbf{W} 其实就是 embedding 参数，在 TensorFlow 中可使用 `tf.get_variable()` 进行定义，然后使用 `tf.nn.embedding_lookup()` 找到那些上下文单次对应的向量 \mathbf{v}_{w_c} .

对 embedding 的解释

$$\underbrace{\begin{bmatrix} W_{1,1}, & \dots, & W_{1,j}, & \dots, & W_{1,D} \\ \dots & & \dots & & \dots \\ W_{k,1}, & \dots, & W_{k,j}, & \dots, & W_{k,D} \\ \dots & & \dots & & \dots \\ W_{K,1}, & \dots, & W_{K,j}, & \dots, & W_{K,D} \end{bmatrix}}_{\mathbf{W} \text{ 的第 } k \text{ 行}} = \underbrace{(W_{k,1}, \dots, W_{k,j}, \dots, W_{k,D})_{1 \times D}}_{\mathbf{W} \text{ 中的第 } k \text{ 行}}$$

$\xrightarrow{\mathbf{x}^T, \text{ 第 } k \text{ 列为 } 1}$

具体例子如图所示

$$\begin{bmatrix} 0 & 0 & 0 & \boxed{1} & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \boxed{10} & \boxed{12} & \boxed{19} \\ 11 & 18 & 25 \end{bmatrix} = [10 \ 12 \ 19]$$

图 148: Embedding lookup in matrix

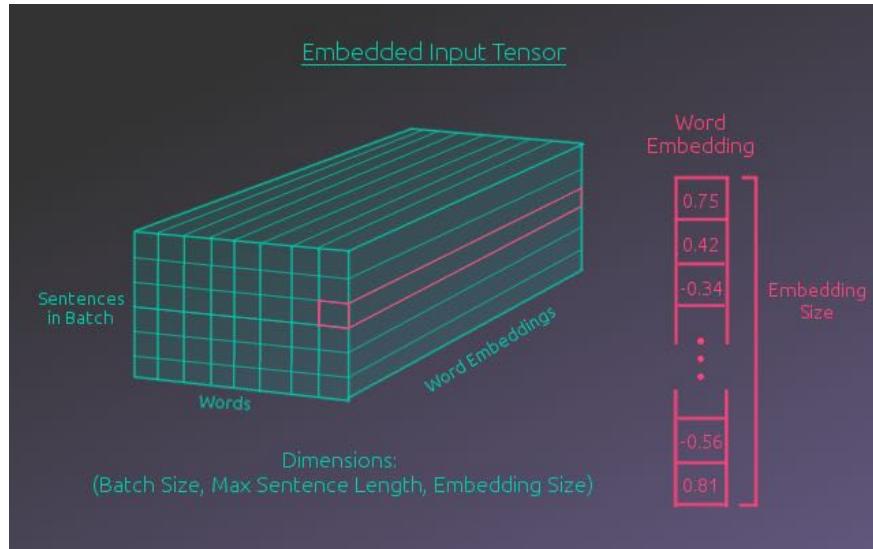


图 149: Batch embedding tensors, cite from Mac Brennan: Neural Translation Model

问题：为何拥有相同上下文的中心词，嵌入向量相似？ 因为拥有相同上下文的词，他们在训练样本中的 target 是相同的，因此在 softmax regression 中被分到相同的类，所以训练得到的嵌入向量就相似。

44.2.2 Skip-Gram Model

One-word target

Dataset & Inference 考虑如下的句子：“The quick brown fox jumps over the lazy dog.”，现设置滑窗大小为 w ，那么会选取当前中心词 context word 之前 w 个词和之后 w 个词（共 $2w$ 个词）作为 target word. 如下左图为 $w = 2$ 时的例子，右图为网络结构

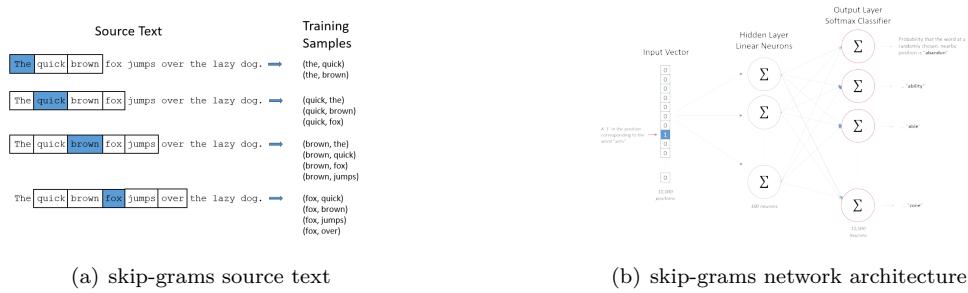


图 150: Decision tree without pruning

Multi-words targets

Inference 结构如图

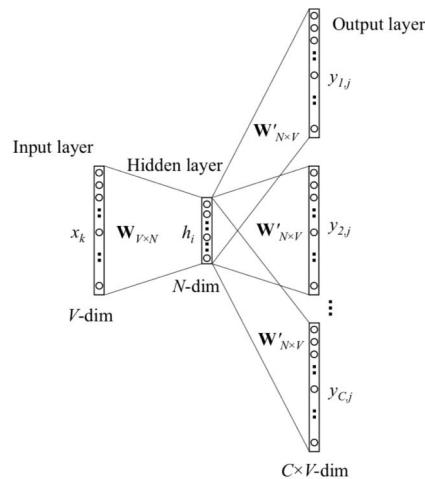


图 151: A Skip-Gram model with multi targets

Objective function 考虑一个训练样本的情况：

$$\begin{aligned}
 & \max p(w_{o1}, w_{o2}, \dots, w_{oc} | w_I) \\
 \Rightarrow & \max \log p(w_{o1}, w_{o2}, \dots, w_{oc} | w_I) \\
 \Rightarrow & \min -\log p(w_{o1}, w_{o2}, \dots, w_{oc} | w_I) \\
 \Rightarrow & \min -\log \prod_{c=1}^C p(w_{oc} | w_I) \quad \because (\text{条件独立假设}) \\
 \Rightarrow & \min -\log \prod_{c=1}^C \frac{e^{u_c}}{\sum_{k=1}^V e^{u_k}} \\
 \Rightarrow & \min -\log \prod_{c=1}^C \frac{e^{\mathbf{v}'_c^\top \mathbf{v}_I}}{\sum_{k=1}^V e^{\mathbf{v}'_k^\top \mathbf{v}_I}} \\
 \Rightarrow & \min -\sum_{c=1}^C u_c + \log \prod_{c=1}^C \left(\sum_{k=1}^V e^{u_k} \right) \\
 \Rightarrow & \min -\sum_{c=1}^C u_c + C \log \left(\sum_{k=1}^V e^{u_k} \right) \\
 \Rightarrow & \min -\sum_{c=1}^C \mathbf{v}'_c^\top \mathbf{v}_I + C \log \left(\sum_{k=1}^V e^{\mathbf{v}'_k^\top \mathbf{v}_I} \right)
 \end{aligned}$$

44.2.3 Subsampling frequent words

考虑句子语料：“The quick brown fox jumps over the lazy dog.”这个例子中存在两个重要的问题：

Source Text	Training Samples
The quick brown fox jumps over the lazy dog.	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog.	(quick, the) (quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog.	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog.	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

图 152: Skip-grams source text

- 从 target word 的角度：对于样本 (fox, the) ，没有什么大的意义，因为“the”这个 target 词出现在几乎每个单词的上下文中
- 从 source word 的角度：对于样本 (the,...) ，样本集合中“the”中心词样本出现过于多的次数，而我们不需要为“the”这个词学太好

因此解决办法是 **Subsampling**，即：

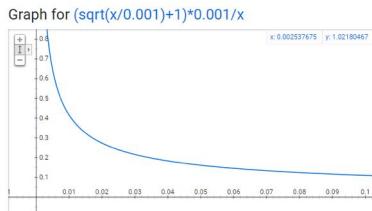


图 153: Subsampling function of word2vec

- 对每个词，统计其在训练语料中的词频 $z(w^{(i)})$ （该词出现次数/总词出现次数）
- 根据词频计算其保留概率

$$p(w^{(i)}) = \left(\sqrt{\frac{z(w^{(i)})}{0.001}} + 1 \right) \cdot \frac{0.001}{z(w^{(i)})}$$

使用这个函数的好处是：

- 实际问题中，没有一个单词（词频）是在语料库中占非常大的比例的，因此我们希望在 x 轴（词频）上查看非常小的值
- 恰好，这个函数就能满足这个需求，比如：
 - $- z(w^{(i)}) \leq 0.0026 \Rightarrow p(w^{(i)}) = 1.0$: 意味着只有词频大于 0.26% 的词才会被采样。

- $z(w^{(i)}) \leq 0.00746 \Rightarrow p(w^{(i)}) = 0.5$: 意味着只有词频等于 0.746% 的词有 50% 的几率保留。
- $z(w^{(i)}) = 1.0 \Rightarrow p(w^{(i)}) = 0.033$: 意味着只有词频等于 100% 的词只有 3.3% 的几率保留。

这样一来：

- 出现频次很少的词，一定被保留，得到充分学习
- 出现频次很多的词，很少被保留，大大缩短时间

44.2.4 Negative sampling

- 将概率的表示用类间相关的 Softmax 函数改为类间独立的 Logistic 函数
- 负采样相当于减少了一次迭代中总分类类别的个数（不再是全部的 $|\mathcal{V}|$ 个）

$$\max \sum_{(l,c) \in \mathcal{D}_p} \log \frac{1}{1 + e^{-\mathbf{v}_c' \mathbf{v}_l}} + \sum_{(l,c) \in \mathcal{D}_n} \log \frac{1}{1 + e^{\mathbf{v}_c' \mathbf{v}_l}}$$

- $\because \frac{1}{1+e^{-\mathbf{v}_c' \mathbf{v}_l}} = \frac{e^{\mathbf{v}_c' \mathbf{v}_l}}{1+e^{\mathbf{v}_c' \mathbf{v}_l}}$ 为发生的概率
- $\frac{1}{1+e^{\mathbf{v}_c' \mathbf{v}_l}} = \therefore 1 - \frac{e^{\mathbf{v}_c' \mathbf{v}_l}}{1+e^{\mathbf{v}_c' \mathbf{v}_l}}$ 为没有发生概率

44.3 BERT (未完成)

44.3.1 Introduction

本节来自论文：[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding\[?\]](#)

44.3.2 Reference

- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding(slides)

44.4 XLNet (未完成)

44.4.1 Introduction

本节来自论文：[XLNet: Generalized Autoregressive Pretraining for Language Understanding\[?\]](#)

45 Text Classification

45.1 Convolutional Neural Networks for Sentence Classification

45.1.1 Introduction

本节来自论文：Convolutional Neural Networks for Sentence Classification[?]

45.1.2 Model formulation

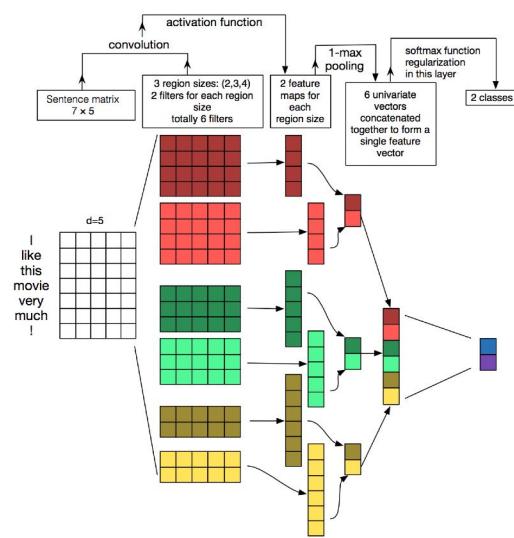


图 154: Architecture of TextCNN[?]

Model inference

45.2 Deep Pyramid Convolutional Neural Networks for Text Categorization

45.2.1 Introduction

本节来自论文：Deep Pyramid Convolutional Neural Networks for Text Categorization[?]

45.2.2 Model formulation

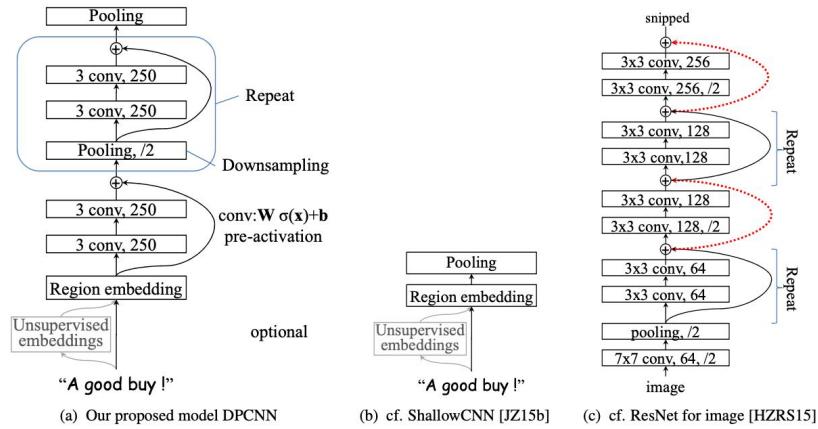


图 155: Architecture of DPCNN[?]

Model inference

46 Semantic Matching

46.1 DSSM

46.1.1 Introduction

本节来自论文：Learning Deep Structured Semantic Models for Web Search using Clickthrough Data (DSSM)[?]

通用的搜索引擎系统语义匹配模型结构包括三个部分：

- 输入层
- 表示层 (BOW/DNN/CNN/RNN...)
- 匹配层

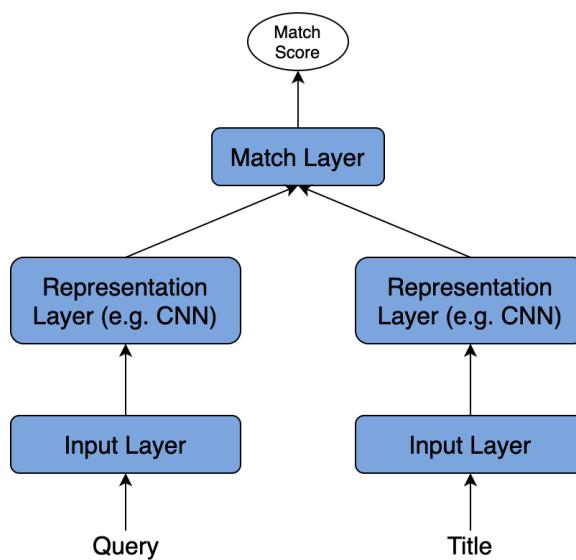


图 156: The structure of deep semantic matching

46.1.2 Model formulation

Model inference 结构如图所示：图中每一步的操作解释如下：

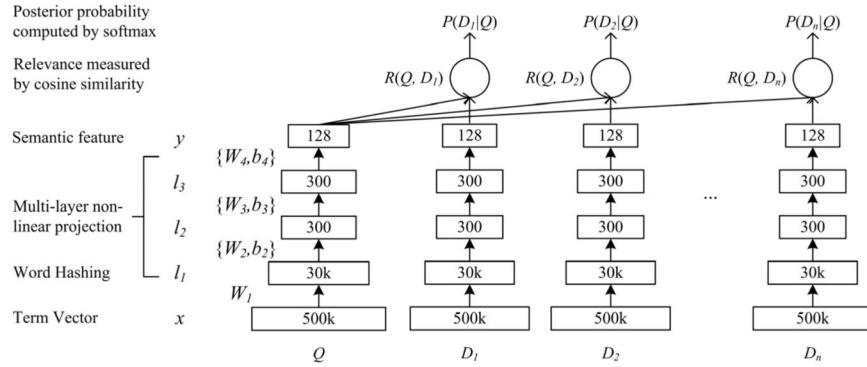


图 157: The structure of DSSM[?]

1. 表示层 $x \Rightarrow l_1$:

$$l_1 = \mathbf{W}_1 x$$

2. 表示层 $l_i \Rightarrow l_{i+1}$:

$$l_{i+1} = f(\mathbf{W}_{i+1} l_i + \mathbf{b}_{i+1}), \quad i \in \{1, \dots, N-1\}$$

3. 表示层 $l_N \Rightarrow y$:

$$y = f(\mathbf{W}_N l_{N-1} + \mathbf{b}_N), \quad (\text{激活函数 } f \text{ 为 tanh})$$

4. 匹配层 $y_Q, y_D \Rightarrow R(Q, D)$:

$$R(Q, D) = \text{cosine}(y_Q, y_D) = \frac{\mathbf{y}_Q^T \mathbf{y}_D}{\|\mathbf{y}_Q\| \|\mathbf{y}_D\|}$$

5. 匹配得分层 $R(Q, D) \Rightarrow P(D | Q)$:

$$P(D | Q) = \frac{\exp(\gamma R(Q, D))}{\sum_{D' \in \mathcal{D}} \exp(\gamma R(Q, D'))} \quad (\text{其中 } \gamma \text{ 为 smoothing factor})$$

47 Machine Reading Comprehension

47.1 NumNet: Machine Reading Comprehension with Numerical Reasoning(未完成)

47.1.1 Introduction

本节选自论文NumNet: Machine Reading Comprehension with Numerical Reasoning[?]

48 Sequence Generation

48.1 SeqGAN (未完成)

48.1.1 Introduction

本文选自SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient[?]

49 Sequence Tagging

49.1 Bidirectional LSTM-CRF Models for Sequence Tagging (Bi-LSTM + CRF) (未完成)

49.1.1 Introduction

本节选自论文[Bidirectional LSTM-CRF Models for Sequence Tagging\[?\]](#)

Part IX

Model Applications of Autonomous Driving

50 Deep Learning Approaches

50.1 A Survey of Deep Learning Techniques for Autonomous Driving

50.1.1 Reference

- A Survey of Deep Learning Techniques for Autonomous Driving[?]

Part X

Model Applications of Real-Time Bidding

51 Deep Reinforcement Learning Approaches

Part XI

Model Applications of Recommender System

52 Low-Rank Matrix Factorization Approaches

参考 Deep Learning for Matching in Search and Recommendation[?]，现代推荐系统包括两部分：

- 召回：粗排，主要使用 collaborative filtering models
- 排序：精排，主要使用 feature based models

Modern RecSys Architecture

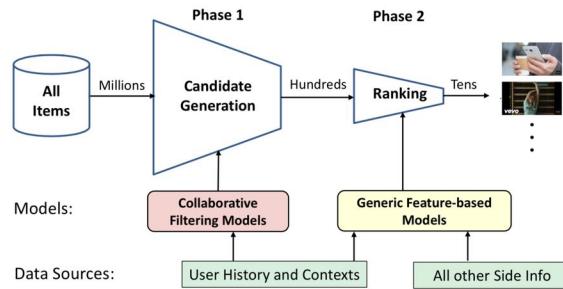


图 158: The architecture of the modern recommender system[?]

52.1 Collaborative Filtering (CF)

52.1.1 Introduction

- 协同过滤是最著名的推荐算法。
 - 前提假设**：一个用户的表现可以通过与他相似的用户进行预测。
- 数学建模：一个矩阵填充问题 (matrix completion problem)。



图 159: Formulate collaborative filtering[?] as matrix completion problem

52.1.2 Model formulation

dataset

- $\mathbf{Y} \in \mathbb{R}^{m \times n}$: 评分矩阵, 其中 m 为总用户的个数, n 为总商品个数

inference

$$\hat{y}_{ui} = b_u + \frac{\sum_{v \in \mathcal{S}^k(u; i)} s_{uv} (y_{vi} - b_v)}{\sum_{v \in \mathcal{S}^k(u; i)} s_{uv}}$$

符号含义如下：

- b_u : 用户 u 本身的偏好
- $\mathcal{S}^k(u; i)$: 用户 u 的 k 个相似用户构成的用户集合, 且这些相似用户在商品 i 上有打分记录
- s_{uv} : 用户 u 和用户 v 的相似度
- y_{ui} : 用户 u 对商品 i 的真实评分
- \hat{y}_{ui} : 用户 u 对商品 i 的预测评分

问题：如何计算用户偏好 b_u , 相似用户集合 $\mathcal{S}^k(u; i)$ 和用户相似度数值 s_{uv} ?

- 用户偏好 b_u ：用户 u 对现有商品评分的均值
- 相似用户集合 $\mathcal{S}^k(u; i)$ ：所有在商品 i 上有评分记录的用户
- 用户相似度数值 s_{uv} ：用户 u 和用户 v 在**两个共同商品评分向量**上计算所得的相似度

52.1.3 Characteristics

- 优点:
 - 易于理解
- 缺点:
 - 计算速度慢
 - 不是一个优化问题
 - 非在线学习
 - 只有记忆性，没有扩展性

52.2 Singular Value Decomposition (SVD)

52.2.1 Introduction

- 奇异值分解 (SVD) 是用于矩阵填充最著名的技术

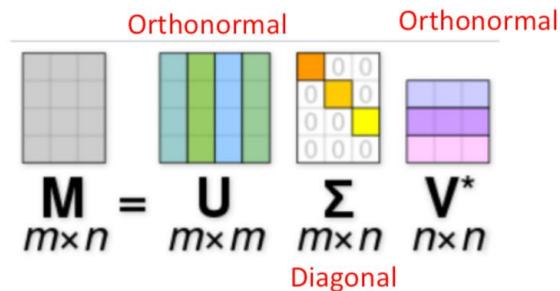


图 160: Singular Value Decomposition (SVD) is the most well-known technique for matrix completion

52.2.2 Model formulation

dataset

- $\mathbf{Y} \in \mathbb{R}^{m \times n}$: 评分矩阵, 其中 m 为总用户的个数, n 为总商品个数

inference

1. 填充评分矩阵 \mathbf{Y} 中的缺失值 (没有评分记录) 为 0
2. 求解一个 SVD 问题 :

$$\begin{aligned} & \min_{\mathbf{U}, \Sigma, \mathbf{V}} (\mathbf{Y} - \mathbf{U}\Sigma\mathbf{V}^T)^2 \\ &= \min_{\mathbf{U}, \Sigma, \mathbf{V}} \sum_{i=1}^m \sum_{j=1}^n (y_{ij} - (\mathbf{U}\Sigma\mathbf{V})_{ij}^T)^2 \end{aligned}$$

3. 只使用矩阵 \mathbf{U} 和 \mathbf{V} 中的 K 维, 从而获得一个 low rank 模型估计 $\hat{\mathbf{Y}}$

52.2.3 Characteristics

- 优点:
 - 将协同过滤建模为一个优化问题
- 缺点:
 - 缺失数据 (0) 和观测数据 (非 0) 拥有相同的权重 (>99% sparsity)
 - 没有正则化项, 模型容易过拟合
 - 训练参数量大
 - 不是迭代的优化学习方式

52.3 Matrix Factorization (MF)

52.3.1 Introduction

- 在推荐系统领域，该算法被称作 Matrix Factorization[?] (MF) 或 SVD：
 - 该模型将任意一个用户和任意一个商品表示为维度相同的隐向量 (latent vector)，也就是所谓的 (**ID embedding**)
 - 一个用户和一个商品的相互作用以向量内积 (**inner product**) 的形式进行建模，内积表示用户的隐偏好和商品的隐属性有多匹配
 - 也可以将该模型使用 cross-entropy 用以二分类问题
- MF 模型利用**ID 建模用户 和商品**

52.3.2 Model formulation

dataset

<label> <user ID> <item ID>

inference

$$\hat{y}_{ui} = \mathbf{p}_u^T \mathbf{q}_i = \sum_{k=1}^K p_{uk} q_{ik}$$

其中超参数 K 为隐向量维度。

理解：

- 隐向量每一个维度 $k \in \{1, \dots, K\}$ 可理解为一个隐类分布点
- p_{uk} 表示用户 u 对隐类 k 的喜好程度
- q_{ik} 表示商品 i 对隐类 k 的所属程度

loss function

$$J = \sum_u \sum_i w_{ui} (y_{ui} - \hat{y}_{ui})^2 + \lambda (\sum_u \|\mathbf{p}_u\|^2 + \sum_i \|\mathbf{q}_i\|^2)$$

52.3.3 Characteristics

- 优点:

- 训练样本简单
- 参数数量少
- 空间消耗小
- 时间消耗小

- 缺点:

- 没有利用用户属性和商品属性
- 没有利用上下文信息

53 Embedding Learning Approaches

53.1 Real-time Personalization using Embeddings for Search Ranking at Airbnb

53.1.1 Introduction

本节来自论文：[Real-time Personalization using Embeddings for Search Ranking at Airbnb\[?\]](#)

该论文来自于 Airbnb 团队，为 KDD2018 最佳论文，不仅是当前业务和机器学习结合最好的体现，也是 I2I 最好的尝试，是对 word2vec 花样玩法的最好诠释。

文章构成 本文针对解决短期推荐和长期推荐应用场景下的问题，提出了两种模型：

- short-term recommendation: Listing Embeddings
- long-term recommendation: User-type & Listing-type Embeddings

学习目的 学习得到每个商品在**用户兴趣空间（类似 nlp 中的语义空间）**的向量表达 $\mathbf{v}_{l_i} \in \mathbb{R}^d$ ，使得相似的商品在向量空间中的距离更近。

53.1.2 Model training for listing embeddings (Short-term)

Dataset

- listing ids l_i : 每一个商品 (Airbnb 业务中为住房信息) 的编号索引
- click session $s = (l_1, \dots, l_M)$
 - 不同用户的点击序列属于不同的 click session；
 - 同一用户两个**连续点击** l_i 和 l_{i+1} 如果时间间隔超过 30 分钟，则认为属于不同 session；
 - 同一用户两个**连续点击** l_i 和 l_{i+1} 如果时间间隔不超过 30 分钟，则认为属于同一 session。
- entire set S : 所有 session s 的集合

Objective(重点) 因为从用户角度讲，出现在一个 session 中的 ids 为相似的，因此用户点击一个 ids l_i ，会希望点击同一 session 中的其他 ids l_{i+j} ，即**最大化条件概率** $\mathbb{P}(l_{i+j}|l_i)$ ，因此目标函数为利用 skip-gram 最大化所有 session 中的条件概率，即：

$$\max \mathcal{L} = \sum_{s \in S} \sum_{l_i \in s} \left(\sum_{-m \leq j \leq m, j \neq 0} \log \mathbb{P}(l_{i+j}|l_i) \right)$$

问题 1：如何定义条件概率 $\mathbb{P}(l_{i+j}|l_i)$ ？ Softmax function

$$\mathbb{P}(l_{i+j}|l_i) = \frac{\exp(\mathbf{v}_{l_i}^T \mathbf{v}'_{l_{i+j}})}{\sum_{l=1}^{|\mathcal{V}|} \exp(\mathbf{v}_{l_i}^T \mathbf{v}'_l)}$$

公式中使用内积 $\mathbf{v}_{l_i}^T \mathbf{v}'_{l_{i+j}}$ 恰好可以衡量两个向量之间的相似度，而我们目标正是最大化这些相关 listings 的相似度。符号表示如下：

- \mathbf{v}_l : listing l 的输入向量表示

- \mathbf{v}_l' : listing l 的输出向量表示
- m : 一个 session 内上下文最大划窗大小
- \mathcal{V} : 词表大小 (所有的 listing ids)

问题 2：词表大小 \mathcal{V} 严重限制了模型的计算效率，如何解决？ Sigmoid function + Negative sampling

- 生成一个 positive pairs (l, c) 的集合 \mathcal{D}_p ，即对于一个 session 内的中心 listing l ，选取划窗范围 m 以内的上下文 listings c
- 生成一个 negative pairs (l, c) 的集合 \mathcal{D}_n ，即从整个词表 \mathcal{V} 中随机抽样 n 个负样本

因此上述原始的优化目标函数变为了：

$$\arg \max_{\theta} \sum_{(l,c) \in \mathcal{D}_p} \log \frac{1}{1 + e^{-\mathbf{v}_c' \mathbf{v}_l}} + \sum_{(l,c) \in \mathcal{D}_n} \log \frac{1}{1 + e^{\mathbf{v}_c' \mathbf{v}_l}}$$

其中待学习参数 θ 为: $\mathbf{v}_l, \mathbf{v}_c, l, c \in V$ 。注意:

- $\because \frac{1}{1 + e^{-\mathbf{v}_c' \mathbf{v}_l}} = \frac{e^{\mathbf{v}_c' \mathbf{v}_l}}{1 + e^{\mathbf{v}_c' \mathbf{v}_l}}$ 为点击的概率
- $\therefore 1 - \frac{e^{\mathbf{v}_c' \mathbf{v}_l}}{1 + e^{\mathbf{v}_c' \mathbf{v}_l}} = \frac{1}{1 + e^{\mathbf{v}_c' \mathbf{v}_l}}$ 为没有被点击的概率
- **目标函数为最大化正样本上的点击概率，同时最大化负样本上没有点击的概率**
- 原始模型采用 softmax 函数，由于 \mathcal{V} 数量过大导致计算效率极其低下，因此这里改用 sigmoid 函数，提高计算效率
- \mathcal{D}_p 和 \mathcal{D}_n 中的 l 都是相同的一个中心 listing，即用于一次更新迭代的一个 batch size 的数据集中， \mathcal{D}_p 和 \mathcal{D}_n 针对唯一的一个中心词 l ，下同

Model Update-1: Booking listing as Global Context Airbnb 团队根据实际业务经验，将 click sessions set S 划分为两类：

- booked session. 以预定作为终结的 session
- exploratory session. 没有以预定作为终结的 session，譬如只是继续浏览

实际业务经验可知，从获取上下文相似性的角度来说，两种 session 都是有用的。但是 booked session 拥有**明确的监督信息**，因此可以更好地调整优化。（**这也是这篇 paper 牛逼的地方，把监督信息融入进了无监督学习任务，从而提高模型效果**）

因此上述目标函数 update 为：

$$\arg \max_{\theta} \sum_{(l,c) \in \mathcal{D}_p} \log \frac{1}{1 + e^{-\mathbf{v}_c' \mathbf{v}_l}} + \sum_{(l,c) \in \mathcal{D}_n} \log \frac{1}{1 + e^{\mathbf{v}_c' \mathbf{v}_l}} + \log \frac{1}{1 + e^{-\mathbf{v}_{l_b}' \mathbf{v}_l}}$$

注：由该式可以得知：

- 实际训练过程中，一个 batch 为一个确定中心词 l 所在 session 子集的相关信息，即：

- 当前 batch 中只存在一个 l
- \mathcal{D}_p 中的 l 和 \mathcal{D}_n 中的 l 是同一个 l
- \mathcal{D}_p 中的 c 为当前 session 中所有 context listings 的子集
- \mathcal{D}_n 中的 c 为在所有 global listings 中对非当前 session 中的 listings 的采样
- l_b 为当前 l 所在 session 的唯一 booking listing
- 这里 $\log \frac{1}{1+e^{-\mathbf{v}'_{l_b} \mathbf{v}_l}}$ 之所以没有求和符号，是因为一个 session 中只存在一个 booking listing

Model Update-2: Adapting Training for Congregated Search 根据实际旅行酒店预定网站的用户使用习惯：

- 用户只会在某一特定 market (譬如：想去的地方) 内进行搜索，因此 \mathcal{D}_p 基本上都是与 l 在同一 market 内的 listings
- 由于随机负采样， \mathcal{D}_n 里基本上都是与 l 不在同一 market 的 listings

实验发现，这样的不平衡将会导致 l 所在 market 内部的相似度次优解 (sub-optimal within-market similarities)，即：

- 不同 market 内的 listing 在向量空间内的位置会有很大区别
- 而同一 market 内的 listing 在向量空间区分效果很差 (因为没有设置相关的负采样进行学习)

因此添加中心 listing l 所在 market 内的负采样集 \mathcal{D}_{m_n}

$$\begin{aligned} \arg \max_{\theta} & \sum_{(l,c) \in \mathcal{D}_p} \log \frac{1}{1+e^{-\mathbf{v}'_c \mathbf{v}_l}} + \sum_{(l,c) \in \mathcal{D}_n} \log \frac{1}{1+e^{\mathbf{v}'_c \mathbf{v}_l}} \\ & + \log \frac{1}{1+e^{-\mathbf{v}'_{l_b} \mathbf{v}_l}} + \sum_{(l,m_n) \in \mathcal{D}_{m_n}} \log \frac{1}{1+e^{\mathbf{v}'_{m_n} \mathbf{v}_l}} \end{aligned}$$

注意 \mathcal{D}_{m_n} 中任意 listing 的要求为：

- 与 l 在同一个 market 内 (譬如：都在北京)
- 不存在于 l 所在的 positive session \mathcal{D}_p
- \mathcal{D}_{m_n} 与 \mathcal{D}_n 的区别在于：
 - \mathcal{D}_n 为全局范围内的负采样集 (global negative sampling)
 - \mathcal{D}_{m_n} 为与中心 listing l 所处同一 market 内的负采样集 (market negative sampling)

Cold start listing embeddings Airbnb 实际业务中每天都会有新的 listing 被创建发布，由于新 listings 没有点击记录，因此不存在于训练样本的 sessions 集合 S 中。因此需要利用已经存在的 listings 的 embedding 为新的 listings 创建 embedding。

如何链接已知 listings 进行创建？ 用户创建时需要提供 meta-data，譬如：location, price, listing type 等，Airbnb 根据 meta-data，找到三个 listings，且满足：

- 半径为 10 英里的范围内距离最近

- 相同的 listing type
- 属于同一个 price bucket

然后将这三个 listings 的 embedding $e_{l_1}, e_{l_2}, e_{l_3}$ 的均值作为新品的 embedding :

$$e_{\text{new}} = \frac{1}{3}(e_{l_1} + e_{l_2} + e_{l_3})$$

即通过利用基于显式向量 (explicit vector) 的特征匹配规则，为新的 listing 找到对应的 (implicit vector)。在 Airbnb 的实际业务中，通过这项技术，Airbnb 可以 cover 住 98% 的新品。

Examining Listing Embeddings Airbnb 团队选择 8 亿个 sessions 作为训练样本，设置嵌入向量的维度为 $d = 32$ ，选择 Model Update-2 对应的模型作为训练模型进行训练。训练结束后，为了评价通过 embeddings 技术获取的 listings 的特征，Airbnb 团队做了如下工作：

- 首先，在学习得到的 embedding 上进行 k-means clustering($k = 100$)，用于检测地理相似度(geographical similarity) 信息是否被编码进 embeddings 中。下图161所示结果表明，所有位于相近 location 的 listings 被聚类进同一簇中，因此结论是 geographical similarity 的信息被有效地编码进了 embeddings 中。

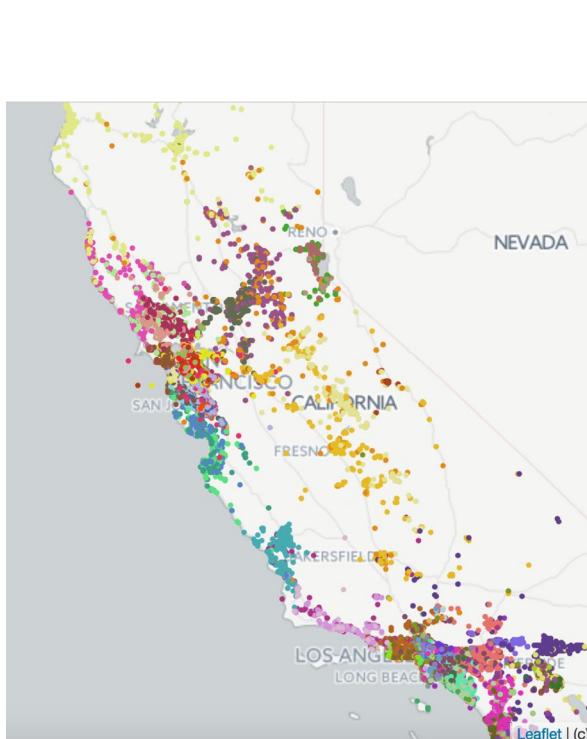


图 161: California Listing Embedding Clusters

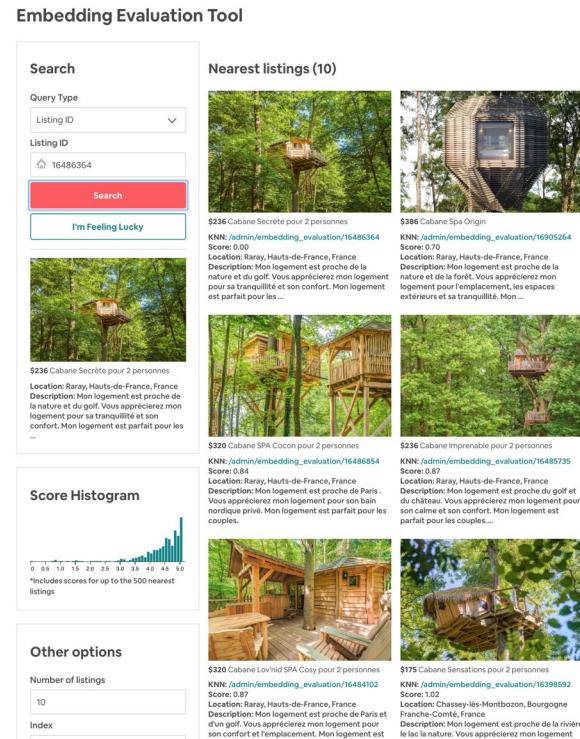


图 162: Embeddings Evaluation Tool

- 其次，对所有来自 Los Angeles 的不同显式特征listing types取值的 listings 和不同显式特征price ranges取值的 listings 计算了 cosine similarities 用于评估 listings 之间的相似性，结果如下，因此可知来自相同 listing type 和相同 price range 的 listings 之间的相似度高于不同 listing type 和不同 price range 的 listings 之间的相似度，因此结论是 listing type 和 price range 的信息被有效编码进了 embeddings 中。

Room Type	Entire Home	Private Room	Shared Room	Price Range	<\$30	\$30-\$60	\$60-\$90	\$90-\$120	\$120+
				<\$30	0.916	0.887	0.882	0.871	0.854
Entire Home	0.895	0.875	0.848	\$30-\$60	0.906	0.889	0.876	0.865	
Private Room		0.901	0.865	\$60-\$90		0.902	0.883	0.880	
Shared Room			0.896	\$90-\$120			0.898	0.890	
				\$120+				0.909	

图 163: Cosine similarities between different Listing Types

图 164: Cosine similarities between different Price Ranges

- 对于其他 meta-data 的属性的检验，譬如 style, price 等，为了方便检验，Airbnb 公司开发了配套的检索工具，用于检索 embedding space 上 knn 返回的 listings，如图 162



图 165: Similar Listings using Embeddings

53.1.3 Model training for user-type & listing-type embeddings (Long-term)

上一节的优化目标：

$$\begin{aligned} \arg \max_{\theta} & \sum_{(l,c) \in \mathcal{D}_p} \log \frac{1}{1 + e^{-\mathbf{v}_c' \mathbf{v}_l}} + \sum_{(l,c) \in \mathcal{D}_n} \log \frac{1}{1 + e^{\mathbf{v}_c' \mathbf{v}_l}} \\ & + \log \frac{1}{1 + e^{-\mathbf{v}'_b \mathbf{v}_l}} + \sum_{(l,m_n) \in \mathcal{D}_{mn}} \log \frac{1}{1 + e^{\mathbf{v}'_{m_n} \mathbf{v}_l}} \end{aligned}$$

因为 \mathcal{D}_p 中任意 session 的生成是根据连续 listings click 的短期行为产生的，所以非常适合找到同一 market 内部的 listings 之间的相似性，因此适合于：

- 短期推荐（因为根据用户搜索习惯，用户会搜索同一 market 内的 listings）
- 展示与用户最近搜索 sessions 中的 listings 相似的 listing

但是平台中存储的大多数数据是用户的长期历史行为，因此自然想到得在模型中使用这些长期历史行为的数据。但是使用什么长期历史行为数据？作者使用了 booking listings 的数据。而针对一个用户的类型会随时间发生变换，同时其对 listing 类型的喜爱也会发生变换，因此，在长期推荐的场景下，将使用 listing-type embedding 和 user-type embedding 用于替代简单的 listing embedding，而 user 和 listing 则是通过基于如下表的规则映射方式找到对应的 user-type 和 listing-type。

Buckets	1	2	3	4	5	6	7	8	Buckets	1	2	3	4	5	6	7	8
Country	US	CA	GB	FR	MX	AU	ES	...	Market	SF	NYC	LA	HK	PHL	AUS	LV	...
Listing Type	Ent	Priv	Share						Language	en	es	fr	jp	ru	ko	de	...
\$ per Night	<40	40-55	56-69	70-83	84-100	101-129	130-189	190+	Device Type	Mac	Msft	Andr	Ipad	Tablet	Iphone		...
\$ per Guest	<21	21-27	28-34	35-42	43-52	53-75	76+		Full Profile	Yes	No						
Num Reviews	0	1	2-5	6-10	11-35	35+			Profile Photo	Yes	No						
Listing 5 Star %	0-40	41-60	61-90	90+					Num Bookings	0	1	2-7	8+				
Capacity	1	2	3	4	5	6+			\$ per Night	<40	40-55	56-69	70-83	84-100	101-129	130-189	190+
Num Beds	1	2	3	4+					\$ per Guest	<21	21-27	28-34	35-42	43-52	53-75	76+	
Num Bedrooms	0	1	2	3	4+				Capacity	<2	2-2.6	2.7-3	3.1-4	4.1-6	6.1+		
Num Bathroom	0	1	2	3+					Num Reviews	<1	1-3.5	3.6-10	>10				
New Guest Acc %	<60	61-90	>91						Listing 5 Star %	0-40	41-60	61-90	90+				
									Guest 5 Star %	0-40	41-60	61-90	90+				

图 166: Mappings of listing meta data to listing type buckets
 图 167: Mappings of user meta data to user type buckets

Training Procedure 因此以训练 user-type embeddings 为中的目标函数为：

$$\arg \max_{\theta} \sum_{(u_t, c) \in \mathcal{D}_{\text{book}}} \log \frac{1}{1 + e^{-v_c' v_{u_t}}} + \sum_{(u_t, c) \in \mathcal{D}_{\text{neg}}} \log \frac{1}{1 + e^{v_c' v_{u_t}}}$$

其中：

- $(u_t, c) \in \mathcal{D}_{\text{book}}$ 中的 c 表示用户类型 u_t 所 booking 过的 listing
- $(u_t, c) \in \mathcal{D}_{\text{neg}}$ 中的 c 表示用户类型 u_t 没有 booking 过的 listing
- 实际训练时，一个 batch 内 $\mathcal{D}_{\text{book}}$ 和 \mathcal{D}_{neg} 中的 u_t 因为其为同一个 user-type

于此同时，以训练 listing-type embeddings 为中的目标函数为：

$$\arg \max_{\theta} \sum_{(l_t, c) \in \mathcal{D}_{\text{book}}} \log \frac{1}{1 + e^{-v_c' v_{l_t}}} + \sum_{(l_t, c) \in \mathcal{D}_{\text{neg}}} \log \frac{1}{1 + e^{v_c' v_{l_t}}}$$

其中：

- $(l_t, c) \in \mathcal{D}_{\text{book}}$ 中的 c 表示在 listing 类型 l_t 上有过 booking 行为的 user-types
- $(l_t, c) \in \mathcal{D}_{\text{neg}}$ 中的 c 表示表示在 listing 类型 l_t 上没有过 booking 行为的 user-types
- 实际训练时，一个 batch 内 $\mathcal{D}_{\text{book}}$ 和 \mathcal{D}_{neg} 中的 l_t 因为其为同一个 listing-type

Explicit Negatives for Rejections 实际上，当 user 选择预定 listing 的时候，listing 的主人会先查看 user 的信息，然后有可能会拒绝将自己的 listing 提供给该用户。所以为了尽量减少用户被拒绝的可能性，设计目标函数的时候会添加进一些历史存储的 rejection events， $(u_t, l_t) \in \mathcal{D}_{\text{rej}}$ 表示 listing-type l_t 对应的 listing 拒绝被预定给 user-type u_t 对应的用户。因此以训练 user-type embeddings 为中的目标函数为：

$$\arg \max_{\theta} \sum_{(u_t, c) \in \mathcal{D}_{\text{book}}} \log \frac{1}{1 + e^{-v_c' v_{u_t}}} + \sum_{(u_t, c) \in \mathcal{D}_{\text{neg}}} \log \frac{1}{1 + e^{v_c' v_{u_t}}} + \sum_{(u_t, l_t) \in \mathcal{D}_{\text{rej}}} \log \frac{1}{1 + e^{v_{l_t}' v_{u_t}}}$$

其中：

- $(u_t, c) \in \mathcal{D}_{\text{book}}$ 中的 c 表示用户类型 u_t 所 booking 过的 listing
- $(u_t, c) \in \mathcal{D}_{\text{neg}}$ 中的 c 表示用户类型 u_t 没有 booking 过的 listing

- $(u_t, l_t) \in \mathcal{D}_{\text{rej}}$ 中的 l_t 表示回绝用户类型 u_t 进行 booking 的 listing，反过来，要是有 u_t 对应的 user 主动拒绝入住的 l_t 数据那就更好了
- 实际训练时，一个 batch 内 $\mathcal{D}_{\text{book}}$, \mathcal{D}_{neg} 和 \mathcal{D}_{rej} 中的 u_t 因对其为同一个 user-type

于此同时，以训练 listing-type embeddings 为中心的目标函数为：

$$\arg \max_{\theta} \sum_{(l_t, c) \in \mathcal{D}_{\text{book}}} \log \frac{1}{1 + e^{-\mathbf{v}'_c \mathbf{v}_{l_t}}} + \sum_{(l_t, c) \in \mathcal{D}_{\text{neg}}} \log \frac{1}{1 + e^{\mathbf{v}'_c \mathbf{v}_{l_t}}} + \sum_{(l_t, u_t) \in \mathcal{D}_{\text{rej}}} \log \frac{1}{1 + e^{\mathbf{v}'_{u_t} \mathbf{v}_{l_t}}}$$

其中：

- $(l_t, c) \in \mathcal{D}_{\text{book}}$ 中的 c 表示在 listing 类型 l_t 上有过 booking 行为的 user-types
- $(l_t, c) \in \mathcal{D}_{\text{neg}}$ 中的 c 表示在 listing 类型 l_t 上没有过 booking 行为的 user-types
- $(l_t, u_t) \in \mathcal{D}_{\text{rej}}$ 中的 u_t 表示被 l_t 明确拒绝 booking 的 user-type
- 实际训练时，一个 batch 内 $\mathcal{D}_{\text{book}}$, \mathcal{D}_{neg} 和 \mathcal{D}_{rej} 中的 l_t 因对其为同一个 listing-type

53.2 Learning and Transferring IDs Representation in E-commerce (未完成)

53.2.1 Introduction

本文选自Learning and Transferring IDs Representation in E-commerce[?]

54 Click Through Rate Prediction Approaches

为何需要点击率预估算法？

- 消费者：信息过载，从海量信息中找到自己感兴趣的信息非常困难
- 生产者：让自己生产的信息脱颖而出，收到广大消费者的关注十分困难
- 平台：按点击收费

$$\begin{aligned} \text{收入} &= \text{广告点击量} \times \text{单次点击价格} \\ &= \underbrace{\text{广告展示量}}_{\text{const}} \times \underbrace{\text{广告点击率}}_{\text{KEY!}} \times \underbrace{\text{单次点击价格}}_{\text{const}} \end{aligned}$$

- “跑马圈地”时代已过，量已达到瓶颈，增速难以为继（获取成本较高）
- “精耕细作”时代已至，靠个性化体验提高转化率，拉起增长

Problem of logistic regression based methods

- predict immediate response
- assume user chooses each item independently

54.1 Factorization Machines (FM)

54.1.1 Introduction

本节来自论文：Factorization Machines[?] 以及 Factorization Machines with libFM[?]

54.1.2 Model formulation

Dataset 推荐系统（尤其在 CTR）领域的数据最大的两个特点：

- Multi-field
- Sparse

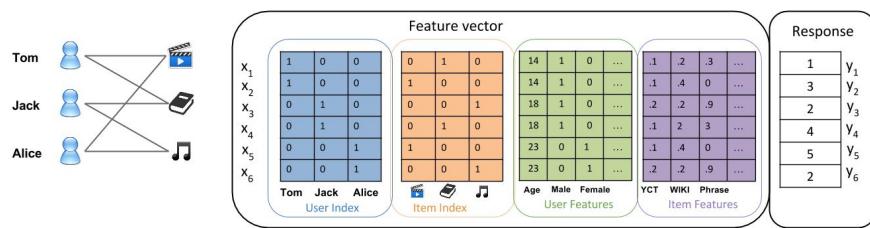


图 168: Multi-field data format for recommender system

Model inference

- simple linear regression model: $\hat{y}(\mathbf{x}) = w_0 + \sum_{j=1}^d w_j x_j$
- two-way interactions polynomial model: $\hat{y}(\mathbf{x}) = w_0 + \sum_{j=1}^d w_j x_j + \sum_{j_1=1}^d \sum_{j_2=j_1}^d w_{j_1, j_2} (x_{j_1} x_{j_2})$

Polynomial-2 Model \Rightarrow Factorization Machines:

- Polynomial-2 Model 优势：
建模二阶特征交叉 (two-way feature-interactions)，而特征交叉是推荐系统领域建模的关键操作。
- Polynomial-2 Model 缺陷：

$$\begin{aligned} \frac{\partial J(\mathbf{w})}{\partial w_{j_1, j_2}} &= \frac{\partial J(\mathbf{w})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{j_1, j_2}} \\ &= \frac{\partial \hat{y}}{\partial w_{j_1, j_2}} x_{j_1} x_{j_2} \end{aligned}$$

所以参数 w_{j_1, j_2} 的迭代更新公式为：

$$w_{j_1, j_2} \leftarrow w_{j_1, j_2} - \eta \frac{\partial J(\mathbf{w})}{\partial \hat{y}} x_{j_1} x_{j_2}$$

因为原始数据的稀疏性，使得 $x_{j_1} x_{j_2}$ 在很大概率上为 0，因此参数 w_{j_1, j_2} 总是得不到更新学习，所以模型效果会很差。

- 解决办法：引入了 Factorization Machines

$$\hat{y} = w_0 + \sum_{j=1}^d w_j x_j + \sum_{j_1=1}^d \sum_{j_2=j_1}^d (\mathbf{v}_{j_1}^\top \cdot \mathbf{v}_{j_2}) x_{j_1} x_{j_2}$$

注意：上述表达式非最佳表示，我们可以对二阶项进行变换从而挖掘出 FM 模型的本质，即：

$$\begin{aligned}
& \sum_{j_1=1}^d \sum_{j_2=j_1+1}^d (\mathbf{v}_{j_1}^\top \cdot \mathbf{v}_{j_2}) x_{j_1} x_{j_2} \\
&= \frac{1}{2} \left(\sum_{j_1=1}^d \sum_{j_2=1}^d (\mathbf{v}_{j_1}^\top \cdot \mathbf{v}_{j_2}) x_{j_1} x_{j_2} - \sum_{j_1=1}^d (\mathbf{v}_{j_1}^\top \cdot \mathbf{v}_{j_1}) x_{j_1} x_{j_1} \right) \quad (\because \mathbf{v}_{j_1}^\top \cdot \mathbf{v}_{j_2} = \sum_{f=1}^F v_{j_1,f} v_{j_2,f}) \\
&= \frac{1}{2} \left(\sum_{j_1=1}^d \sum_{j_2=1}^d \left(\sum_{f=1}^F v_{j_1,f} v_{j_2,f} \right) x_{j_1} x_{j_2} - \sum_{j_1=1}^d \left(\sum_{f=1}^F v_{j_1,f} v_{j_1,f} \right) x_{j_1} x_{j_1} \right) \quad (\text{展开内部所有括号}) \\
&= \frac{1}{2} \left(\sum_{j_1=1}^d \sum_{j_2=1}^d \sum_{f=1}^F v_{j_1,f} v_{j_2,f} x_{j_1} x_{j_2} - \sum_{j_1=1}^d \sum_{f=1}^F v_{j_1,f} v_{j_1,f} x_{j_1} x_{j_1} \right) \\
&= \frac{1}{2} \sum_{f=1}^F \underbrace{\left(\left(\sum_{j_1=1}^d v_{j_1,f} x_{j_1} \right) \left(\sum_{j_2=1}^d v_{j_2,f} x_{j_2} \right) - \sum_{j_1=1}^d v_{j_1,f}^2 x_{j_1}^2 \right)}_{\text{最关键一步，将特征值的交叉分离}} \\
&= \frac{1}{2} \sum_{f=1}^F \left(\left(\sum_{j=1}^d v_{j,f} x_j \right)^2 - \sum_{j=1}^d v_{j,f}^2 x_j^2 \right)
\end{aligned}$$

因此最终 Factorization Machines 模型可以表示为：

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{j=1}^d w_j x_j + \frac{1}{2} \sum_{f=1}^F \left(\left(\sum_{j=1}^d v_{j,f} x_j \right)^2 - \sum_{j=1}^d v_{j,f}^2 x_j^2 \right)$$

Loss function 我们以二分类问题为例进行推导：

- 内函数： $z(\mathbf{x}) = w_0 + \sum_{j=1}^d w_j x_j + \frac{1}{2} \sum_{f=1}^F \left(\left(\sum_{j=1}^d v_{j,f} x_j \right)^2 - \sum_{j=1}^d v_{j,f}^2 x_j^2 \right)$
- 外函数： $\hat{y} = \phi(z) = \frac{1}{1+\exp(-z)}$

目标函数为：

$$\min J(\mathbf{w}, \mathbf{v}) = -\frac{1}{N} \sum_{i=1}^N (y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})) + \lambda \sum_{j=1}^d w_j^2 + \lambda \sum_{j=1}^d \sum_{f=1}^F v_{j,f}^2$$

Optimization 我们以随机梯度下降（一条训练样本）为例进行损失函数部分的推导：

$$\therefore \frac{\partial J(\mathbf{x}; y; \mathbf{w}; \mathbf{v})}{\partial \theta} = \underbrace{\frac{\partial J}{\partial \phi(z)} \frac{\partial \phi(z)}{\partial z} \frac{\partial z}{\partial \theta}}_{\text{公共部分可先求出}}$$

$$\frac{\partial J}{\partial \phi(z)} = \frac{\partial}{\partial \phi(z)} - (y \log \phi(z) + (1-y) \log(1-\phi(z)))$$

$$\begin{aligned}
&= - \left(\frac{y}{\phi(z)} + \frac{1-y}{1-\phi(z)} \underbrace{(-1)}_{\text{易忽略}} \right) \\
&= - \left(\frac{y}{\phi(z)} + \frac{y-1}{1-\phi(z)} \right) \quad (\text{注意一定要保留负号在外边不动和分母不变}) \\
&= - \frac{y(1-\phi(z)) + \phi(z)(y-1)}{\phi(z)(1-\phi(z))} \\
&= - \frac{y-y\phi(z) + y\phi(z) - \phi(z)}{\phi(z)(1-\phi(z))} \\
&= - \frac{y-\phi(z)}{\phi(z)(1-\phi(z))} \\
\therefore \frac{\partial \phi(z)}{\partial z} &= \frac{\partial}{\partial z} \left(\frac{1}{1+e^{-z}} \right) \\
&= - \frac{1}{(1+e^{-z})^2} e^{-z} (-1) \\
&= \frac{e^{-z}}{1+e^{-z}} \frac{1}{1+e^{-z}} \\
&= \frac{1+e^{-z}-1}{1+e^{-z}} \frac{1}{1+e^{-z}} \\
&= \left(1 - \frac{1}{1+e^{-z}} \right) \frac{1}{1+e^{-z}} \quad (\text{注意 } \phi(z) = \frac{1}{1+e^{-z}}) \\
&= (1-\phi(z))\phi(z) \\
\therefore \frac{\partial J}{\partial \phi(z)} \frac{\partial \phi(z)}{\partial z} &= - \frac{y-\phi(z)}{\phi(z)(1-\phi(z))} (1-\phi(z))\phi(z) = -(y-\phi(z))
\end{aligned}$$

下边只需再计算 $\frac{\partial z}{\partial \theta}$ ，即：

$$\frac{\partial z}{\partial \theta} = \begin{cases} 1, & \text{if } \theta = w_0 \\ x_j, & \text{if } \theta = w_j \\ x_j \sum_{j=1}^d v_{j,f} x_j - x_j^2 v_{j,f}, & \text{if } \theta = v_{j,f} \end{cases}$$

其中对于 $v_{j,f}$ 的求偏导过程如下：

$$\begin{aligned}
\frac{\partial z}{\partial v_{j,f}} &= \frac{\partial}{\partial v_{j,f}} \left(\frac{1}{2} \sum_{f=1}^F \left(\left(\sum_{j=1}^d v_{j,f} x_j \right)^2 - \sum_{j=1}^d v_{j,f}^2 x_j^2 \right) \right) \\
&= \frac{\partial}{\partial v_{j,f}} \left(\frac{1}{2} \left(\sum_{j=1}^d v_{j,f} x_j \right)^2 - \frac{1}{2} \sum_{j=1}^d v_{j,f}^2 x_j^2 \right) \\
&= \frac{1}{2} \cdot 2 \left(\sum_{j=1}^d v_{j,f} x_j \right) x_j - \frac{1}{2} x_j^2 \cdot (2v_{j,f}) \\
&= x_j \sum_{j=1}^d v_{j,f} x_j - x_j^2 v_{j,f}
\end{aligned}$$

因此由参数的迭代更新公式可知 FM 的优势为：

- 每个参数的更新只取决于它本身的维度 j ，即我命由我，自强独立
- 可以与未知的另一特征进行交叉，即做好自己，随时交互

54.1.3 Model implement

Dataset 考虑到实际推荐中数据的特性

- 结构性
- 离散性
- 多域性
- 稀疏性

我们对数据做图示 LibSVM 格式的存储: 代码如下:

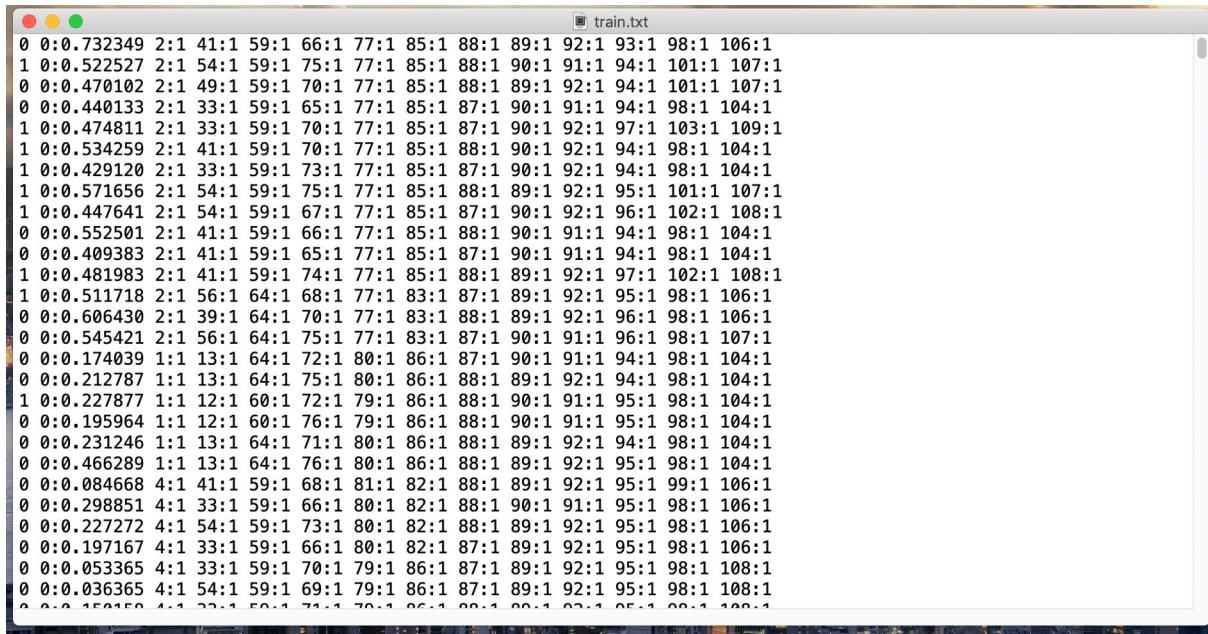


图 169: Data libsvm format

```
#####
Created on 2019/04/08 01:31
author: Tong Jia
email: cecilio.jia@gmail.com
description:
An implementation of Factorization Machines (FM).
See algorithm and hyperparameter details:
[Rendle 2010] (https://www.csie.ntu.edu.tw/~b97053/paper/Rendle2010FM.pdf)
The algorithm is developed with TensorFlow Estimator based on TensorFlow 1.12.0 version.
#####

import tensorflow as tf
from tensorflow.contrib.layers import xavier_initializer, l2_regularizer
import os

FLAGS = tf.flags.FLAGS
# -----Hyperparameters of estimator-----
```

```

tf.flags.DEFINE_enum(name="phase",
                     default=None,
                     enum_values=["train", "eval", "predict", "export"],
                     help="A string, representing the phase of estimator")
tf.flags.DEFINE_string(name="model_dir",
                      default=None,
                      help="A string, representing the folder path of saved model files")
tf.flags.DEFINE_string(name="export_dir",
                      default=None,
                      help="A string, representing the basic folder path of export .pb files")
# -----Hyperparameters of estimator (optional)-----
tf.flags.DEFINE_boolean(name="perform_valid_during_train",
                        default=True,
                        help="(optional) A boolean, instructing whether to perform validation on valid dataset
                             during train phase")
tf.flags.DEFINE_integer(name="log_step_count_steps",
                       default=500,
                       help="(optional) An integer, representing the frequency, in number of global steps, that
                             the global step/sec will be
                             logged during training",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="save_checkpoints_steps",
                       default=20000,
                       help="(optional) An integer, representing save checkpoints every this many steps",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="keep_checkpoint_max",
                       default=2,
                       help="(optional) An integer, representing max number of saved model checkpoints",
                       lower_bound=1,
                       upper_bound=None)
# -----Hyperparameters of input function-----
tf.flags.DEFINE_string(name="data_dir",
                      default=None,
                      help="A string, representing the folder path of dataset")
tf.flags.DEFINE_string(name="delimiter",
                      default=None,
                      help="A string, separating consecutive <index j>:<value j> pairs in a line of data file")
tf.flags.DEFINE_string(name="separator",
                      default=None,
                      help="A string, separating feature index(left part) and feature value(right part) in a
                           specific pair")
tf.flags.DEFINE_integer(name="batch_size",
                       default=None,
                       help="An integer, representing the number of consecutive elements of this dataset to
                             combine in a single batch",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="epochs",
                       default=None,
                       help="An integer, representing the number of times the dataset should be repeated",

```

```

        lower_bound=1,
        upper_bound=None)
# -----Hyperparameters of input function (optional)-----
tf.flags.DEFINE_boolean(name="shuffle",
                        default=True,
                        help="(optional) A boolean, instructing whether to randomly shuffle the samples of
                              training dataset")
tf.flags.DEFINE_integer(name="buffer_size",
                       default=100000,
                       help="(optional) An integer scalar, denoting the number of bytes to buffer",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="num_parallel_calls",
                       default=4,
                       help="(optional) An integer scalar, representing the number elements to process in
                             parallel",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_boolean(name="use_dtype_high_precision",
                       default=False,
                       help="(optional) A boolean, instructing the dtype of both input function and model
                             function. If False, use
                             tf.float32; if True, use
                             tf.float64")
tf.flags.DEFINE_string(name="name_feat_inds",
                      default="inds",
                      help="(optional) A string, representing the name of feature indices in return dict")
tf.flags.DEFINE_string(name="name_feat_vals",
                      default="vals",
                      help="(optional) A string, representing the name of feature values in return dict")
# -----Hyperparameters of model function-----
tf.flags.DEFINE_string(name="task",
                      default=None,
                      help="A string, representing the type of task (binary or regression)")
tf.flags.DEFINE_integer(name="field_size",
                       default=None,
                       help="An integer scalar, representing the number of fields(number of columns before
                             one-hot encoding) of dataset")
tf.flags.DEFINE_integer(name="feat_size",
                       default=None,
                       help="An integer scalar, representing the number of features(number of columns after
                             one-hot encoding) of dataset")
tf.flags.DEFINE_integer(name="embed_size",
                       default=None,
                       help="An integer scalar, representing the dimension of embedding vectors for all
                             features")
# -----Hyperparameters of model function (optional)-----
tf.flags.DEFINE_boolean(name="use_global_bias",
                       default=True,
                       help="(optional) A boolean, instructing whether to use global bias in model inference")
tf.flags.DEFINE_boolean(name="use_linear",
                       default=True,

```

```

        help="(optional) A boolean, instructing whether to use linear part in model inference")
tf.flags.DEFINE_float(name="lamb",
                      default=0.001,
                      help="(optional) A float scalar, representing the coefficient of regularization term",
                      lower_bound=0.0,
                      upper_bound=None)
tf.flags.DEFINE_string(name="optimizer",
                      default="adam",
                      help="(optional) A string, representing the type of optimizer")
tf.flags.DEFINE_float(name="learning_rate",
                      default=0.003,
                      help="(optional) A float scalar, representing the learning rate of optimizer",
                      lower_bound=0.0,
                      upper_bound=None)

def input_fn(filenames,
             delimiter,
             separator,
             batch_size,
             epochs,
             shuffle=True,
             buffer_size=100000,
             num_parallel_calls=4,
             dtype=tf.float32,
             name_feat_inds="inds",
             name_feat_vals="vals"):

    """
    The input function for loading multi-field sparse dataset. The columns are
    separated by argument delimiter(e.g. " ") with the following schema (libsvm format):
    <label> <index 1>:<value 1> ... <index j>:<value j> ... <index d>:<value d> (d is the field size of dataset)
    e.g.
    0 0:0.732349 2:1 41:1 59:1 66:1 77:1 85:1 88:1 89:1 92:1 93:1 98:1 106:1
    where:
    delimiter is always " " in txt file format, "," in csv file format;
    separator is always equal to ":".
    Note:
    1. The input function is only used for dataset with one-hot active value in each field.
    2. In each line, the order of all fields must be fixed.
    3. The input function can be used for binary classification and regression task:
       binary classification: <label> in {0, 1};
       regression: <label> in (-inf, inf).
    Parameters
    -----
    :param filenames: list
        A list of string, containing one or more paths of filenames.
    :param delimiter: str
        A str, separating consecutive <index j>:<value j> pairs in data files.
    :param separator: str
        A str, separating feature index(left part of key-value pair) and feature value(right part of key-value
        pair) in one specific pair.
    :param batch_size: int

```

An integer scalar, representing the number of consecutive elements of this dataset to combine in a single batch.

:param epochs: int
An integer scalar, representing the number of times the dataset should be repeated.

:param shuffle: bool, optional
A boolean(defaults to True), instructing whether to randomly shuffle the elements of this dataset.

:param buffer_size: int, optional
An integer scalar(defaults to 100000), denoting the number of bytes to buffer.

:param num_parallel_calls: int, optional
An integer scalar(defaults to 4), representing the number elements to process in parallel.

:param dtype: tf.Dtype, optional
A tf.DType(defaults to tf.float32), representing the numeric type of values. it always takes value from [tf.float32, tf.float64].

:param name_feat_inds: str, optional
A string, representing the name of feature indices in return dict.

:param name_feat_vals: str, optional
A string, representing the name of feature values in return dict.

Returns

:return: dict
A dict of two Tensors, representing features(including feature indices and feature values) in a single batch.

```
{
    <name_feat_inds>: tf.Tensor of feature indices in shape of (None, field_size),
    <name_feat_vals>: tf.Tensor of feature values in shape of (None, field_size)
}
```

:return: Tensor
A Tensor in shape of (None), representing labels in a single batch.

"""
def map_func(line):
 columns = tf.string_split(source=[line], delimiter=delimiter, skip_empty=False).values
 label = tf.string_to_number(string_tensor=columns[0], out_type=dtype)
 splits = tf.string_split(source=columns[1:], delimiter=separator, skip_empty=False)
 feats = tf.reshape(
 tensor=splits.values, shape=splits.dense_shape
) # A tensor in shape of (field_size, 2), the first column contains feature indices, the second column
 # contains feature values
 inds, vals = tf.split(value=feats, num_or_size_splits=2, axis=1) # Two tensors in shape of (field_size,
 1)
 inds = tf.reshape(tensor=tf.string_to_number(string_tensor=inds, out_type=tf.int32), shape=[-1]) # A
 # tensor in shape of (field_size)
 vals = tf.reshape(tensor=tf.string_to_number(string_tensor=vals, out_type=dtype), shape=[-1]) # A
 # tensor in shape of (field_size)
 return {name_feat_inds: inds, name_feat_vals: vals}, label

dataset = tf.data.TextLineDataset(filenames=filenames, buffer_size=buffer_size). \
 map(map_func=map_func, num_parallel_calls=num_parallel_calls)
if (shuffle ==True):
 dataset = dataset.shuffle(buffer_size=buffer_size)
dataset = dataset. \
 repeat(count=epochs). \

```

batch(batch_size=batch_size, drop_remainder=False)
dataset = dataset.prefetch(buffer_size=1)
iterator = dataset.make_one_shot_iterator()
batch_feats, batch_labels = iterator.get_next()
return batch_feats, batch_labels

def model_fn(features, labels, mode, params):
    """Model function of DeepFM for predictive analytics of high dimensional sparse data.

    Args of dict params:
        task: str
            A string, representing the type of task.
            Note:
                it must take value from ["binary", "regression"];
                it instruct the type of loss function:
                    "binary": sigmoid cross-entropy;
                    "regression": mean squared error.
        field_size: int
            An integer scalar, representing the number of fields(number of columns before one-hot encoding) of
            dataset.
        feat_size: int
            An integer scalar, representing the number of features(number of columns after one-hot encoding) of
            dataset.
        embed_size: int
            An integer scalar, representing the dimension of embedding vectors for all features.
        use_global_bias: bool
            A boolean, instructing whether to use global bias in model inference.
        use_linear: bool
            A boolean, instructing whether to use linear part in model inference.
        lamb: float
            A float scalar, representing the coefficient of regularization term (the larger the value of lamb,
            the stronger the penalty is).
        optimizer: str
            A string, representing the type of optimizer.
        learning_rate: float
            A float scalar, representing the learning rate of optimizer.
        dtype: tf.Dtype
            A tf.DType, representing the numeric type of values.
            Note:
                it must take value from [tf.float32, tf.float64];
                it must be consistent with <dtype> of input function.
        name_feat_inds: str
            A string, representing the name of feature indices in return dict.
            Note:
                it must be consistent with <name_feat_inds> of input function.
        name_feat_vals: str
            A string, representing the name of feature values in return dict.
            Note:
                it must be consistent with <name_feat_vals> of input function.
        reuse: bool
            A boolean, which takes value from [False, True, tf.AUTO_REUSE].

```

```

seed: int or None
    If integer scalar, representing the random seed of tensorflow;
    If None, random choice.

"""
# -----Declare all hyperparameters from params-----
task =params["task"]
field_size =params["field_size"]
feat_size =params["feat_size"]
embed_size =params["embed_size"]
use_global_bias =params["use_global_bias"]
use_linear =params["use_linear"]
lamb =params["lamb"]
optimizer =params["optimizer"]
learning_rate =params["learning_rate"]
dtype =params["dtype"]
name_feat_inds =params["name_feat_inds"]
name_feat_vals =params["name_feat_vals"]
reuse =params["reuse"]
seed =params["seed"]
# -----Hyperparameters for threshold for binary classification task
threshold =0.5
# -----Hyperparameters for exponential decay(*manual optional*)-----
decay_steps =5000
decay_rate =0.998
staircase =True
# -----Hyperparameters for information showing-----
name_probability_output ="prob"
name_classification_output ="class"
name_regression_output ="pred"
value_error_warning_task ="Argument of model function <task>: \"{}\" is invalid. It must be in
                                [\"binary\", \"regression\"]".format(task)
value_error_warning_optimizer ="Argument value of <optimizer>: {} is not supported.".format(optimizer)

if seed !=None:
    tf.set_random_seed(seed)

# -----Build model inference-----
with tf.variable_scope(name_or_scope="inference", reuse=reuse):
    with tf.name_scope(name="inputs"):
        ids =features[name_feat_inds]
        x =features[name_feat_vals]

    with tf.name_scope(name="global-bias"):
        b =tf.get_variable(name="b",
                           shape=[1],
                           dtype=dtype,
                           initializer=tf.zeros_initializer(dtype=dtype),
                           regularizer=l2_regularizer(scale=lamb),
                           trainable=use_global_bias)

    with tf.name_scope(name="lr-part"):
        W =tf.get_variable(name="W",

```

```

        shape=[feat_size],
        dtype=dtype,
        initializer=xavier_initializer(uniform=True, dtype=dtype), # *manual optional*
        regularizer=l2_regularizer(scale=lamb),
        trainable=use_linear)

# -----embedding lookup op for first order weights-----
w = tf.nn.embedding_lookup(params=W, ids=ids) # A tensor in shape of (None, field_size)
ylr =tf.reduce_sum(input_tensor=tf.multiply(x=x, y=w), # A tensor in shape of (None, field_size)
                    axis=1,
                    keepdims=False) # A tensor in shape of (None)

with tf.name_scope(name="fm-part"):

    V = tf.get_variable(name="V",
                         shape=[feat_size, embed_size],
                         dtype=dtype,
                         initializer=xavier_initializer(uniform=False, dtype=dtype), # manual optional
                         regularizer=None, # *manual optional*
                         trainable=True)

# -----embedding lookup op for second order weights-----
v = tf.nn.embedding_lookup(params=V, ids=ids) # A tensor in shape of (None, field_size, embed_size)
xreshape =tf.expand_dims(input=x, axis=-1) # A tensor in shape of (None, field_size, 1)
vx =tf.multiply(x=xreshape, y=v) # A tensor in shape of (None, field_size, embed_size)
p = tf.square(x=tf.reduce_sum(input_tensor=vx, axis=1, keepdims=False)) # a tensor in shape of
                           (None, embedding_size)
q = tf.reduce_sum(input_tensor=tf.square(x=vx), # A tensor in shape of (None, field_size,
                  embedding_size)
                  axis=1,
                  keepdims=False) # A tensor in shape of (None, embedding_size)
yfm =0.5 *tf.reduce_sum(input_tensor=tf.subtract(x=p, y=q), # A tensor in shape of (None,
                           embedding_size)
                           axis=1,
                           keepdims=False) # A tensor in shape of (None)

with tf.name_scope(name="output"):

    if use_global_bias ==False and use_linear ==False:
        logits =yfm
    elif use_global_bias ==False and use_linear ==True:
        logits =ylr +yfm
    elif use_global_bias ==True and use_linear ==False:
        logits =b +yfm
    else:
        logits =b +ylr +yfm

    if task == "binary":
        logits =tf.squeeze(input=logits, axis=1) # A tensor in shape of (None)
        probs =tf.nn.sigmoid(x=logits)
        classes =tf.cast(x=tf.greater(x=tf.nn.sigmoid(x=logits), y=threshold), dtype=tf.int32)
        predictions ={  

            name_probability_output: probs,  

            name_classification_output: classes  

        }

```

```

        elif task == "regression":
            logits = tf.squeeze(input=logits, axis=1) # A tensor in shape of (None)
            predictions = {
                name_regression_output: logits
            }
        else:
            raise ValueError(value_error_warning_task)

# -----Provide an estimator spec for `ModeKeys.PREDICTION` mode-----
if mode == tf.estimator.ModeKeys.PREDICT:
    export_outputs = {
        tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
            tf.estimator.export.PredictOutput(outputs=predictions)
    } # For usage of tensorflow serving
    return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions, export_outputs=export_outputs)

# -----Build loss function-----
if task == "binary":
    loss = tf.reduce_mean(input_tensor=tf.nn.sigmoid_cross_entropy_with_logits(labels=labels,
                                                                           logits=logits),
                           axis=0,
                           keepdims=False) # A scalar, representing the training loss of current batch training
                           dataset
elif task == "regression":
    loss = tf.reduce_mean(input_tensor=tf.square(x=tf.subtract(x=labels, y=logits)),
                           axis=0,
                           keepdims=False) # A scalar, representing the training loss of current batch training
                           dataset
else:
    raise ValueError(value_error_warning_task)

reg = tf.reduce_sum(input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
                     axis=0,
                     keepdims=False,
                     name="regularization") # A scalar, representing the regularization loss of current batch
                     training dataset
loss += reg

# -----Provide an estimator spec for `ModeKeys.EVAL` mode-----
if mode == tf.estimator.ModeKeys.EVAL:
    if task == "binary":
        eval_metric_ops = {
            "accuracy": tf.metrics.accuracy(labels=labels,
                                             predictions=predictions[name_classification_output]),
            "precision": tf.metrics.precision(labels=labels,
                                              predictions=predictions[name_classification_output]),
            "recall": tf.metrics.recall(labels=labels, predictions=predictions[name_classification_output]),
            "auc": tf.metrics.auc(labels=labels, predictions=predictions[name_classification_output])
        }
    elif task == "regression":
        eval_metric_ops = {
            "rmse": tf.metrics.root_mean_squared_error(labels=labels,
}

```

```

                predictions=predictions[name_regression_output])

        }

    else:
        raise ValueError(value_error_warning_task)

    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, predictions=predictions,
                                      eval_metric_ops=eval_metric_ops)

# -----Build optimizer-----
global_step =tf.train.get_or_create_global_step(graph=tf.get_default_graph()) # Define a global step for
                                training step counter

if optimizer == "sgd":
    opt_op =tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
elif optimizer == "sgd-exp-decay":
    decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)
    opt_op =tf.train.GradientDescentOptimizer(learning_rate=decay_learning_rate)
elif optimizer == "momentum":
    opt_op =tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                       momentum=0.9,
                                       use_nesterov=False)
elif optimizer == "momentum-exp-decay":
    decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)
    opt_op =tf.train.MomentumOptimizer(learning_rate=decay_learning_rate,
                                       momentum=0.9,
                                       use_nesterov=False)
elif optimizer == "nesterov":
    opt_op =tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                       momentum=0.9,
                                       use_nesterov=True)
elif optimizer == "nesterov-exp-decay":
    decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)
    opt_op =tf.train.MomentumOptimizer(learning_rate=decay_learning_rate,
                                       momentum=0.9,
                                       use_nesterov=True)
elif optimizer == "adagrad":
    opt_op =tf.train.AdagradOptimizer(learning_rate=learning_rate,
                                      initial_accumulator_value=0.1)
elif optimizer == "adadelta":
    opt_op =tf.train.AdadeltaOptimizer(learning_rate=learning_rate,
                                       rho=0.95)
elif optimizer == "rmsprop":

```

```

    opt_op =tf.train.RMSPropOptimizer(learning_rate=learning_rate,
                                      decay=0.9)

    elif optimizer == "adam":
        opt_op =tf.train.AdamOptimizer(learning_rate=learning_rate,
                                       beta1=0.9,
                                       beta2=0.999)

    else:
        raise NotImplementedError(value_error_warning_optimizer)

    train_op =opt_op.minimize(loss=loss, global_step=global_step, name="train_op")

    # -----Provide an estimator spec for `ModeKeys.TRAIN` mode-----
    if (mode ==tf.estimator.ModeKeys.TRAIN):
        return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

def main(unused_argv):
    # -----Declare all hyperparameters of terminal interface-----
    phase =FLAGS.phase
    perform_valid_during_train =FLAGS.perform_valid_during_train # (optional) Note: if use validation dataset
                                                               during train phase, the valid set and train set are
                                                               in same data_dir
    log_step_count_steps =FLAGS.log_step_count_steps # (optional)
    save_checkpoints_steps =FLAGS.save_checkpoints_steps # (optional)
    keep_checkpoint_max =FLAGS.keep_checkpoint_max # (optional)
    model_dir =FLAGS.model_dir
    export_dir =FLAGS.export_dir
    data_dir =FLAGS.data_dir
    delimiter =FLAGS.delimiter
    separator =FLAGS.separator
    batch_size =FLAGS.batch_size
    epochs =FLAGS.epochs
    shuffle =FLAGS.shuffle
    buffer_size =FLAGS.buffer_size # (optional)
    num_parallel_calls =FLAGS.num_parallel_calls # (optional)
    use_dtype_high_precision =FLAGS.use_dtype_high_precision # (optional)
    name_feat_inds =FLAGS.name_feat_inds # (optional)
    name_feat_vals =FLAGS.name_feat_vals # (optional)
    task =FLAGS.task
    field_size =FLAGS.field_size
    feat_size =FLAGS.feat_size
    embed_size =FLAGS.embed_size
    use_global_bias =FLAGS.use_global_bias # (optional)
    use_linear =FLAGS.use_linear # (optional)
    lamb =FLAGS.lamb
    optimizer =FLAGS.optimizer
    learning_rate =FLAGS.learning_rate

    PREFIX_TRAIN_FILE ="train"
    PREFIX_EVAL_FILE ="eval"
    PREFIX_PREDICT_FILE ="predict"
    REUSE =False

```

```

SEED =None

if use_dtype_high_precision ==False:
    dtype =tf.float32
else:
    dtype =tf.float64

hparams ={
    "task": task,
    "field_size": field_size,
    "feat_size": feat_size,
    "embed_size": embed_size,
    "use_global_bias": use_global_bias,
    "use_linear": use_linear,
    "lamb": lamb,
    "optimizer": optimizer,
    "learning_rate": learning_rate,
    "dtype": dtype,
    "name_feat_inds": name_feat_inds,
    "name_feat_vals": name_feat_vals,
    "reuse": REUSE,
    "seed": SEED
}
config =tf.estimator.RunConfig(
    log_step_count_steps=log_step_count_steps,
    save_checkpoints_steps=save_checkpoints_steps,
    keep_checkpoint_max=keep_checkpoint_max
)
estimator =tf.estimator.Estimator(model_fn=model_fn, model_dir=model_dir, params=hparams, config=config)
if phase == "train":
    filenames_train =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_TRAIN_FILE + "*"))
    if perform_valid_during_train ==True:
        filenames_valid =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_EVAL_FILE + "*"))
        train_spec =tf.estimator.TrainSpec(input_fn=lambda :input_fn(filenames=filenames_train,
                                                                     delimiter=delimiter,
                                                                     separator=separator,
                                                                     batch_size=batch_size,
                                                                     epochs=epochs,
                                                                     shuffle=shuffle,
                                                                     buffer_size=buffer_size,
                                                                     num_parallel_calls=num_parallel_calls,
                                                                     dtype=dtype,
                                                                     name_feat_inds=name_feat_inds,
                                                                     name_feat_vals=name_feat_vals),
                                            max_steps=None)
        eval_spec =tf.estimator.EvalSpec(input_fn=lambda :input_fn(filenames=filenames_valid,
                                                                    delimiter=delimiter,
                                                                    separator=separator,
                                                                    batch_size=batch_size,
                                                                    epochs=1,
                                                                    shuffle=False,
                                                                    buffer_size=buffer_size,

```

```

        num_parallel_calls=num_parallel_calls,
        dtype=dtype,
        name_feat_inds=name_feat_inds,
        name_feat_vals=name_feat_vals),
        steps=None,
        start_delay_secs=120,
        throttle_secs=600)
    tf.estimator.train_and_evaluate(estimator=estimator, train_spec=train_spec, eval_spec=eval_spec)
else:
    estimator.train(input_fn=lambda : input_fn(filenames=filenames_train,
                                                delimiter=delimiter,
                                                separator=separator,
                                                batch_size=batch_size,
                                                epochs=epochs,
                                                shuffle=shuffle,
                                                buffer_size=buffer_size,
                                                num_parallel_calls=num_parallel_calls,
                                                dtype=dtype,
                                                name_feat_inds=name_feat_inds,
                                                name_feat_vals=name_feat_vals),
                    steps=None,
                    max_steps=None)
elif phase == "eval":
    filenames_eval =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_EVAL_FILE + "*"))
    estimator.evaluate(input_fn=lambda : input_fn(filenames=filenames_eval,
                                                delimiter=delimiter,
                                                separator=separator,
                                                batch_size=batch_size,
                                                epochs=1,
                                                shuffle=False,
                                                buffer_size=buffer_size,
                                                num_parallel_calls=num_parallel_calls,
                                                dtype=dtype,
                                                name_feat_inds=name_feat_inds,
                                                name_feat_vals=name_feat_vals))
elif phase == "predict":
    filenames_predict =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_PREDICT_FILE + "*"))
    p = estimator.predict(input_fn=lambda : input_fn(filenames=filenames_predict,
                                                delimiter=delimiter,
                                                separator=separator,
                                                batch_size=batch_size,
                                                epochs=1,
                                                shuffle=False,
                                                buffer_size=buffer_size,
                                                num_parallel_calls=num_parallel_calls,
                                                dtype=dtype,
                                                name_feat_inds=name_feat_inds,
                                                name_feat_vals=name_feat_vals))
    # -----Usage demo, still need to be accomplished-----
    for ele in p:
        print(ele)
elif phase == "export":
```

```
features ={  
    name_feat_inds: tf.placeholder(dtype=tf.int32, shape=[None, field_size], name=name_feat_inds),  
    name_feat_vals: tf.placeholder(dtype=dtype, shape=[None, field_size], name=name_feat_vals)  
}  
serving_input_receiver_fn =tf.estimator.export.build_raw_serving_input_receiver_fn(features=features)  
estimator.export_savedmodel(export_dir_base=export_dir,  
                           serving_input_receiver_fn=serving_input_receiver_fn)  
else:  
    raise NotImplementedError("Argument <phase> value: {} is not supported.".format(phase))  
  
if __name__ == '__main__':  
    tf.logging.set_verbosity(v=tf.logging.INFO)  
    tf.app.run(main=main)
```

54.1.4 Reference

- Factorization Machines
- Warsaw Data Science - Factorization Machines Introduction

54.2 Field-aware Factorization Machines (FFM)

54.2.1 Introduction

本节来自论文：Field-aware Factorization Machines for CTR Prediction[?]

54.2.2 Model formulation

Model inference 考虑如下一个例子：

Clicked.	Publisher (P)	Advertiser (A)	Gender (G)
Yes	ESPN	Nike	Male

FM 模型的二阶交叉部分为：

$$\phi_{\text{FM}}(\mathbf{x}, \mathbf{w}) = \mathbf{w}_{\text{ESPN}} \cdot \mathbf{w}_{\text{Nike}} + \mathbf{w}_{\text{ESPN}} \cdot \mathbf{w}_{\text{Male}} + \mathbf{w}_{\text{Nike}} \cdot \mathbf{w}_{\text{Male}}$$

FM 存在的问题分析如下：

- 每个特征 (feature) 只有一个隐向量 (latent vector) 去学习和其他所有特征的相互作用 (比如 \mathbf{w}_{ESPN} 不仅要和 \mathbf{w}_{Nike} 交叉，还要和 \mathbf{w}_{Male} 交叉)
- Nike 属于 Advertiser 域，Male 属于 Gender 域，因此 (ESPN, Nike) 和 (ESPN, Male) 的作用是不同的
- 如果 ESPN 特征和其他所有 field 的特征交叉时使用相同的 latent vector，那么会带来**隐向量污染**的问题

因此 FFM[?] 对 FM 的改进方法是：

- 一个特征拥有多个 latent vector ($F - 1$ 个)
- 一个特征的一个 latent vector 只服务于其他某一特定的 field

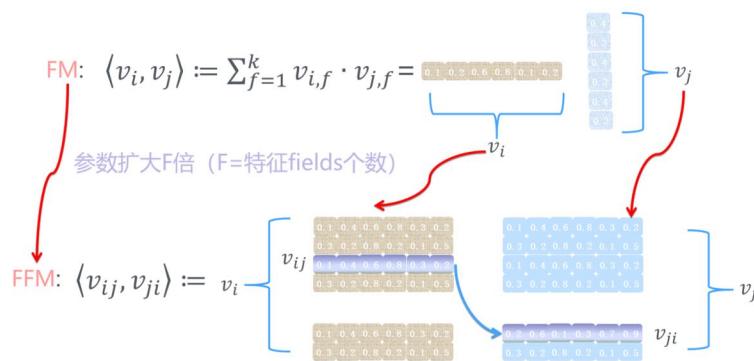


图 170: FM \Rightarrow FFM

因此 FFM 模型的二阶交叉部分为：

$$\phi_{\text{FFM}}(\mathbf{x}, \mathbf{w}) = \mathbf{w}_{\text{ESPN,A}} \cdot \mathbf{w}_{\text{Nike,P}} + \mathbf{w}_{\text{ESPN,G}} \cdot \mathbf{w}_{\text{Male,P}} + \mathbf{w}_{\text{Nike,G}} \cdot \mathbf{w}_{\text{Male,A}}$$

利用数学表达式表示（只考虑两个 field 交叉的项）即：

$$\sum_{j_1=1}^d \sum_{j_2=j_1+1}^d (\mathbf{w}_{j_1, f_2} \cdot \mathbf{w}_{j_2, f_1}) x_{j_1} x_{j_2}$$

说明：

- f_1, f_2 : 维度 j_1 和 j_2 各自对应的 fields
- 假定 fields 的个数为 F, feature 的维度为 D, embedding 的维度为 K, 那么二阶交叉项的参数个数为 $D \times K \times (F - 1)$
- 因此，每个特征 j 都有一个用于和别的某一特定 field 进行交叉的隐向量

Loss function 以 Binary Classification 问题为例，损失函数为：

$$J(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N \left(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right)$$

Optimization 以随机梯度下降（一个训练样本）为例进行推导

$$\begin{aligned} -\nabla_{\mathbf{w}_{j_1, f_2}} J(\mathbf{w}) &= \frac{y}{\hat{y}} \nabla_{\mathbf{w}_{j_1, f_2}} \hat{y} + \frac{y-1}{1-\hat{y}} \nabla_{\mathbf{w}_{j_1, f_2}} \hat{y} \\ &= \frac{y(1-\hat{y}) + \hat{y}(y-1)}{\hat{y}(1-\hat{y})} \nabla_{\mathbf{w}_{j_1, f_2}} \hat{y} \\ &= \frac{y-\hat{y}}{\hat{y}(1-\hat{y})} \nabla_{\mathbf{w}_{j_1, f_2}} \hat{y} \\ \because \nabla_{\mathbf{w}_{j_1, f_2}} \hat{y} &= \frac{\partial \hat{y}}{\partial z} \nabla_{\mathbf{w}_{j_1, f_2}} z \\ &= (1 - \phi(z)) \phi'(z) \nabla_{\mathbf{w}_{j_1, f_2}} z \\ &= (1 - \phi(z)) \phi'(z) \mathbf{w}_{j_2, f_1} x_{j_1} x_{j_2} \\ &= \hat{y}(1 - \hat{y}) x_{j_1} x_{j_2} \mathbf{w}_{j_2, f_1} \\ \therefore -\nabla_{\mathbf{w}_{j_1, f_2}} J(\mathbf{w}) &= (y - \hat{y}) x_{j_1} x_{j_2} \mathbf{w}_{j_2, f_1} \end{aligned}$$

因此参数的更新公式（以 SGD 为例）为：

$$\begin{aligned} \mathbf{w}_{j_2, f_1} &\leftarrow \mathbf{w}_{j_2, f_1} + \eta \left(\frac{1}{m} \sum_{i=1}^m ((y^{(i)} - \hat{y}^{(i)}) x_{j_1}^{(i)} x_{j_2}^{(i)} \mathbf{w}_{j_2, f_1}) + \lambda \mathbf{w}_{j_2, f_1} \right) \\ \mathbf{w}_{j_1, f_2} &\leftarrow \mathbf{w}_{j_1, f_2} + \eta \left(\frac{1}{m} \sum_{i=1}^m ((y^{(i)} - \hat{y}^{(i)}) x_{j_1}^{(i)} x_{j_2}^{(i)} \mathbf{w}_{j_1, f_2}) + \lambda \mathbf{w}_{j_1, f_2} \right) \end{aligned}$$

Notes

- **优势：**
 - 效果好于 Factorization Machines 模型
- **缺陷：**
 - 参数扩大 $F(\text{number of fields})$ 倍

- 大规模使用困难：参数量过大，太耗内存
- 未来可展开工作：
 - 改造 FFM 模型，使得效果接近于 FFM，但是内存消耗远小于 FFM

54.2.3 Model implement

```

"""
Created on 2019/03/31 20:03
author: Tong Jia
email: cecilio.jia@gmail.com
description:
    An implementation of Field-aware Factorization Machines for CTR Prediction (FFM).
    See algorithm and hyperparameter details:
        [Juan et al., 2016] (https://www.csie.ntu.edu.tw/~cjlin/papers/ffm.pdf)
    The algorithm is developed with TensorFlow Estimator based on TensorFlow 1.12.0 version.
"""

import tensorflow as tf
from tensorflow.contrib.layers import xavier_initializer, l2_regularizer
import os

FLAGS = tf.flags.FLAGS
# -----Hyperparameters of estimator-----
tf.flags.DEFINE_enum(name="phase",
                     default=None,
                     enum_values=["train", "eval", "predict", "export"],
                     help="A string, representing the phase of estimator")
tf.flags.DEFINE_string(name="model_dir",
                      default=None,
                      help="A string, representing the folder path of saved model files")
tf.flags.DEFINE_string(name="export_dir",
                      default=None,
                      help="A string, representing the basic folder path of export .pb files")
# -----Hyperparameters of estimator (optional)-----
tf.flags.DEFINE_boolean(name="perform_valid_during_train",
                       default=True,
                       help="(optional) A boolean, instructing whether to perform validation on valid dataset
                             during train phase")
tf.flags.DEFINE_integer(name="log_step_count_steps",
                       default=500,
                       help="(optional) An integer, representing the frequency, in number of global steps, that
                             the global step/sec will be
                             logged during training",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="save_checkpoints_steps",
                       default=20000,
                       help="(optional) An integer, representing save checkpoints every this many steps",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="keep_checkpoint_max",

```

```

        default=2,
        help="(optional) An integer, representing max number of saved model checkpoints",
        lower_bound=1,
        upper_bound=None)
# -----Hyperparameters of input function-----
tf.flags.DEFINE_string(name="data_dir",
                       default=None,
                       help="A string, representing the folder path of dataset")
tf.flags.DEFINE_string(name="delimiter",
                       default=None,
                       help="A string, separating consecutive <index j>:<value j> pairs in a line of data file")
tf.flags.DEFINE_string(name="separator",
                       default=None,
                       help="A string, separating feature index(left part) and feature value(right part) in a
                             specific pair")
tf.flags.DEFINE_integer(name="batch_size",
                       default=None,
                       help="An integer, representing the number of consecutive elements of this dataset to
                             combine in a single batch",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="epochs",
                       default=None,
                       help="An integer, representing the number of times the dataset should be repeated",
                       lower_bound=1,
                       upper_bound=None)
# -----Hyperparameters of input function (optional)-----
tf.flags.DEFINE_boolean(name="shuffle",
                        default=True,
                        help="(optional) A boolean, instructing whether to randomly shuffle the samples of
                              training dataset")
tf.flags.DEFINE_integer(name="buffer_size",
                       default=100000,
                       help="(optional) An integer scalar, denoting the number of bytes to buffer",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="num_parallel_calls",
                       default=4,
                       help="(optional) An integer scalar, representing the number elements to process in
                             parallel",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_boolean(name="use_dtype_high_precision",
                       default=False,
                       help="(optional) A boolean, instructing the dtype of both input function and model
                             function. If False, use
                             tf.float32; if True, use
                             tf.float64")
tf.flags.DEFINE_string(name="name_feat_inds",
                      default="inds",
                      help="(optional) A string, representing the name of feature indices in return dict")
tf.flags.DEFINE_string(name="name_feat_vals",

```

```

        default="vals",
        help="(optional) A string, representing the name of feature values in return dict")
# -----Hyperparameters of model function-----
tf.flags.DEFINE_string(name="task",
                       default=None,
                       help="A string, representing the type of task (binary or regression)")

tf.flags.DEFINE_integer(name="field_size",
                       default=None,
                       help="An integer scalar, representing the number of fields(number of columns before
                             one-hot encoding) of dataset")

tf.flags.DEFINE_integer(name="feat_size",
                       default=None,
                       help="An integer scalar, representing the number of features(number of columns after
                             one-hot encoding) of dataset")

tf.flags.DEFINE_integer(name="embed_size",
                       default=None,
                       help="An integer scalar, representing the dimension of embedding vectors for all
                             features")

# -----Hyperparameters of model function (optional)-----
tf.flags.DEFINE_boolean(name="use_global_bias",
                        default=True,
                        help="(optional) A boolean, instructing whether to use global bias in model inference")

tf.flags.DEFINE_boolean(name="use_linear",
                       default=True,
                       help="(optional) A boolean, instructing whether to use linear part in model inference")

tf.flags.DEFINE_float(name="lamb",
                      default=0.001,
                      help="(optional) A float scalar, representing the coefficient of regularization term",
                      lower_bound=0.0,
                      upper_bound=None)

tf.flags.DEFINE_string(name="optimizer",
                      default="adam",
                      help="(optional) A string, representing the type of optimizer")

tf.flags.DEFINE_float(name="learning_rate",
                      default=0.003,
                      help="(optional) A float scalar, representing the learning rate of optimizer",
                      lower_bound=0.0,
                      upper_bound=None)

def input_fn(filenames,
             delimiter,
             separator,
             epochs,
             batch_size,
             shuffle=True,
             buffer_size=100000,
             num_parallel_calls=4,
             dtype=tf.float32,
             name_feat_inds="inds",
             name_feat_vals="vals"):

    """
    The input function for loading multi-field sparse dataset. The columns are

```

```

separated by argument delimiter(e.g. " ") with the following schema (libsvm format):
<label> <index 1>:<value 1> ... <index j>:<value j> ... <index d>:<value d> (d is the field size of dataset)
e.g.
0 0:0.732349 2:1 41:1 59:1 66:1 77:1 85:1 88:1 89:1 92:1 93:1 98:1 106:1
where:
delimiter is always " " in txt file format, "," in csv file format;
separator is always equal to ":".

Note:
1. The input function is only used for dataset with one-hot active value in each field;
2. In each line, the order of all fields must be fixed;
3. The value of each field can't be empty;
4. The input function can be used for binary classification and regression task:
binary classification: <label> in {0, 1};
regression: <label> in (-inf, inf).

Parameters
-----
:param filenames: list
    A list of string, containing one or more paths of filenames.
:param delimiter: str
    A str, separating consecutive <index j>:<value j> pairs in data files.
:param separator: str
    A str, separating feature index(left part of key-value pair) and feature value(right part of key-value
    pair) in one specific pair.
:param batch_size: int
    An integer scalar, representing the number of consecutive elements of this dataset to combine in a
    single batch.
:param epochs: int
    An integer scalar, representing the number of times the dataset should be repeated.
:param shuffle: bool, optional
    A boolean(defaults to True), instructing whether to randomly shuffle the elements of this dataset.
:param buffer_size: int, optional
    An integer scalar(defaults to 100000), denoting the number of bytes to buffer.
:param num_parallel_calls: int, optional
    An integer scalar(defaults to 4), representing the number elements to process in parallel.
:param dtype: tf.Dtype, optional
    A tf.DType(defaults to tf.float32), representing the numeric type of values. it always takes value from
    [tf.float32, tf.float64].
:param name_feat_inds: str, optional
    A string, representing the name of feature indices in return dict.
:param name_feat_vals: str, optional
    A string, representing the name of feature values in return dict.

Returns
-----
:return: dict
    A dict of two Tensors, representing features(including feature indices and feature values) in a single
    batch.
{
    <name_feat_inds>: tf.Tensor of feature indices in shape of (None, field_size),
    <name_feat_vals>: tf.Tensor of feature values in shape of (None, field_size)
}

```

```

:returns: Tensor
    A Tensor in shape of (None), representing labels in a single batch.
"""

def map_func(line):
    columns = tf.string_split(source=[line], delimiter=delimiter, skip_empty=False).values
    label = tf.string_to_number(string_tensor=columns[0], out_type=dtype)
    splits = tf.string_split(source=columns[1:], delimiter=separator, skip_empty=False)
    feats = tf.reshape(
        tensor=splits.values, shape=splits.dense_shape
    ) # A tensor in shape of (field_size, 2), the first column contains feature indices, the second column
        # contains feature values
    inds, vals = tf.split(value=feats, num_or_size_splits=2, axis=1) # Two tensors in shape of (field_size,
        1)
    inds = tf.reshape(tensor=tf.string_to_number(string_tensor=inds, out_type=tf.int32), shape=[-1]) # A
        # tensor in shape of (field_size)
    vals = tf.reshape(tensor=tf.string_to_number(string_tensor=vals, out_type=dtype), shape=[-1]) # A
        # tensor in shape of (field_size)
    return {name_feat_inds: inds, name_feat_vals: vals}, label

dataset = tf.data.TextLineDataset(filenames=filenames, buffer_size=buffer_size). \
    map(map_func=map_func, num_parallel_calls=num_parallel_calls)
if (shuffle == True):
    dataset = dataset.shuffle(buffer_size=buffer_size)
dataset = dataset.repeat(count=epochs).batch(batch_size=batch_size, drop_remainder=False)
dataset = dataset.prefetch(buffer_size=1)
iterator = dataset.make_one_shot_iterator()
batch_feats, batch_labels = iterator.get_next()
return batch_feats, batch_labels

def model_fn(features, labels, mode, params):
    """Model function of FFM for predictive analytics of high dimensional sparse data.

    Args of dict params:
        task: str
            A string, representing the type of task.
        Note:
            it must take value from ["binary", "regression"];
            it instruct the type of loss function:
            "binary": sigmoid cross-entropy;
            "regression": mean squared error.
        field_size: int
            An integer scalar, representing the number of fields(number of columns before one-hot encoding) of
            dataset.
        feat_size: int
            An integer scalar, representing the number of features(number of columns after one-hot encoding) of
            dataset.
        embed_size: int
            An integer scalar, representing the dimension of embedding vectors for all features.
        use_global_bias: bool
            A boolean, instructing whether to use global bias in model inference.
        use_linear: bool
    """

```

```

    A boolean, instructing whether to use linear part in model inference.

lamb: float
    A float scalar, representing the coefficient of regularization term (the larger the value of lamb,
    the stronger the penalty is).

optimizer: str
    A string, representing the type of optimizer.

learning_rate: float
    A float scalar, representing the learning rate of optimizer.

dtype: tf.Dtype
    A tf.DType, representing the numeric type of values.

Note:
    it must take value from [tf.float32, tf.float64];
    it must be consistent with <dtype> of input function.

name_feat_inds: str
    A string, representing the name of feature indices in return dict.

Note:
    it must be consistent with <name_feat_inds> of input function.

name_feat_vals: str
    A string, representing the name of feature values in return dict.

Note:
    it must be consistent with <name_feat_vals> of input function.

reuse: bool
    A boolean, which takes value from [False, True, tf.AUTO_REUSE].
seed: int or None
    If integer scalar, representing the random seed of tensorflow;
    If None, random choice.

"""
# -----Declare all hyperparameters from params-----
task =params["task"]
field_size =params["field_size"]
feat_size =params["feat_size"]
embed_size =params["embed_size"]
use_global_bias =params["use_global_bias"]
use_linear =params["use_linear"]
lamb =params["lamb"]
optimizer =params["optimizer"]
learning_rate =params["learning_rate"]
dtype =params["dtype"]
name_feat_inds =params["name_feat_inds"]
name_feat_vals =params["name_feat_vals"]
reuse =params["reuse"]
seed =params["seed"]
# -----Hyperparameters for threshold for binary classification task
threshold =0.5
# -----Hyperparameters for exponential decay(*manual optional*)-----
decay_steps =5000
decay_rate =0.998
staircase =True
# -----Hyperparameters for information showing-----
name_probability_output ="prob"
name_classification_output ="class"
name_regression_output ="pred"

```

```

value_error_warning_task ="Argument of model function <task>: \"{}\" is not supported. It must be in " \
                         "[\"binary\", \"regression\"]".format(task)
value_error_warning_optimizer ="Argument value of <optimizer>: {} is not supported.".format(optimizer)

if seed !=None:
    tf.set_random_seed(seed)

# -----Build model inference-----
with tf.variable_scope(name_or_scope="inference", reuse=reuse):
    with tf.name_scope(name="inputs"):
        ids =features[name_feat_inds] # A tensor in shape of (None, field_size)
        x = features[name_feat_vals] # A tensor in shape of (None, field_size)

    with tf.name_scope(name="global-bias"):
        b = tf.get_variable(name="b",
                            shape=[1],
                            dtype=dtype,
                            initializer=tf.zeros_initializer(dtype=dtype),
                            regularizer=l2_regularizer(scale=lamb),
                            trainable=use_global_bias)

    with tf.name_scope(name="lr-part"):
        W = tf.get_variable(name="W",
                            shape=[feat_size],
                            dtype=dtype,
                            initializer=xavier_initializer(uniform=True, dtype=dtype), # *manual optional*
                            regularizer=l2_regularizer(scale=lamb),
                            trainable=use_linear)

        # -----embedding lookup op for first order weights-----
        w = tf.nn.embedding_lookup(params=W, ids=ids) # A tensor in shape of (None, field_size)
        ylr =tf.reduce_sum(input_tensor=tf.multiply(x=x, y=w), # A tensor in shape of (None, field_size)
                           axis=1,
                           keepdims=False) # A tensor in shape of (None)

    with tf.name_scope(name="ffm-part"):
        V = tf.get_variable(name="V",
                            shape=[feat_size, field_size, embed_size],
                            dtype=dtype,
                            initializer=xavier_initializer(uniform=False, dtype=dtype),
                            regularizer=None)

        embedding =tf.nn.embedding_lookup(params=V, ids=ids) # A tensor in shape of (None, field_size,
                                                               field_size, embed_size)

        yffm =tf.constant(value=1, dtype=dtype, shape=[1])
        for i in range(field_size):
            # -----for each valid feature-----
            for j in range(i +1, field_size):
                # -----for each field j to be interacted with feature i-----
                key =embedding[:, i, j, :]# A tensor in shape of (None, embed_size)
                query =embedding[:, j, i, :]# A tensor in shape of (None, embed_size)
                weight =tf.multiply(x=key, y=query) # A tensor in shape of (None, embed_size)
                xixj =tf.expand_dims(input= tf.multiply(x=x[:, i], y=x[:, j]), axis=-1) # A tensor in shape
                                                               of (None, 1)

```

```

        value =tf.reduce_sum(input_tensor=tf.multiply(x=weight, y=xixj), # A tensor in shape of
                             (None, embed_size)
                             axis=-1,
                             keepdims=False) # A tensor in shape of (None)

        yffm +=value

    with tf.name_scope(name="output"):
        if use_global_bias ==False and use_linear ==False:
            logits =yffm
        elif use_global_bias ==False and use_linear ==True:
            logits =ylr +yffm
        elif use_global_bias ==True and use_linear ==False:
            logits =b +yffm
        else:
            logits =b +ylr +yffm

        if task == "binary":
            logits =tf.squeeze(input=logits, axis=1) # A tensor in shape of (None)
            probs =tf.nn.sigmoid(x=logits)
            classes =tf.cast(x=tf.greater(x=tf.nn.sigmoid(x=logits), y=threshold), dtype=tf.int32)
            predictions ={  

                name_probability_output: probs,  

                name_classification_output: classes  

            }
        elif task == "regression":
            logits =tf.squeeze(input=logits, axis=1) # A tensor in shape of (None)
            predictions ={  

                name_regression_output: logits  

            }
        else:
            raise ValueError(value_error_warning_task)

# -----Provide an estimator spec for `ModeKeys.PREDICTION` mode-----
if mode ==tf.estimator.ModeKeys.PREDICT:
    export_outputs ={  

        tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:  

            tf.estimator.export.PredictOutput(outputs=predictions)  

    } # For usage of tensorflow serving
    return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions, export_outputs=export_outputs)

# -----Build loss function-----
if task == "binary":
    loss =tf.reduce_mean(input_tensor=tf.nn.sigmoid_cross_entropy_with_logits(labels=labels,
                                                                           logits=logits),
                           axis=0,
                           keepdims=False) # A scalar, representing the training loss of current batch training
                           dataset
elif task == "regression":
    loss =tf.reduce_mean(input_tensor=tf.square(x=tf.subtract(x=labels, y=logits)),
                           axis=0,
                           keepdims=False) # A scalar, representing the training loss of current batch training
                           dataset

```

```

else:
    raise ValueError(value_error_warning_task)

reg = tf.reduce_sum(input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
                     axis=0,
                     keepdims=False,
                     name="regularization") # A scalar, representing the regularization loss of current batch
                                         training dataset
loss += reg

# -----Provide an estimator spec for `ModeKeys.EVAL` mode-----
if mode ==tf.estimator.ModeKeys.EVAL:
    if task =="binary":
        eval_metric_ops ={ 
            "accuracy": tf.metrics.accuracy(labels=labels,
                                              predictions=predictions[name_classification_output]),
            "precision": tf.metrics.precision(labels=labels,
                                              predictions=predictions[name_classification_output]),
            "recall": tf.metrics.recall(labels=labels, predictions=predictions[name_classification_output]),
            "auc": tf.metrics.auc(labels=labels, predictions=predictions[name_classification_output])
        }
    elif task =="regression":
        eval_metric_ops ={ 
            "rmse": tf.metrics.root_mean_squared_error(labels=labels,
                                              predictions=predictions[name_regression_output])
        }
    else:
        raise ValueError(value_error_warning_task)
return tf.estimator.EstimatorSpec(mode=mode, loss=loss, predictions=predictions,
                                   eval_metric_ops=eval_metric_ops)

# -----Build optimizer-----
global_step =tf.train.get_or_create_global_step(graph=tf.get_default_graph()) # Define a global step for
                                         training step counter
if optimizer =="sgd":
    opt_op =tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
elif optimizer =="sgd-exp-decay":
    decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)
    opt_op =tf.train.GradientDescentOptimizer(learning_rate=decay_learning_rate)
elif optimizer =="momentum":
    opt_op =tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                       momentum=0.9,
                                       use_nesterov=False)
elif optimizer =="momentum-exp-decay":
    decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,

```

```

                staircase=staircase)
opt_op =tf.train.MomentumOptimizer(learning_rate=decay_learning_rate,
                                    momentum=0.9,
                                    use_nesterov=False)

elif optimizer == "nesterov":
    opt_op =tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                    momentum=0.9,
                                    use_nesterov=True)

elif optimizer == "nesterov-exp-decay":
    decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)
    opt_op =tf.train.MomentumOptimizer(learning_rate=decay_learning_rate,
                                    momentum=0.9,
                                    use_nesterov=True)

elif optimizer == "adagrad":
    opt_op =tf.train.AdagradOptimizer(learning_rate=learning_rate,
                                    initial_accumulator_value=0.1)

elif optimizer == "adadelta":
    opt_op =tf.train.AdadeltaOptimizer(learning_rate=learning_rate,
                                    rho=0.95)

elif optimizer == "rmsprop":
    opt_op =tf.train.RMSPropOptimizer(learning_rate=learning_rate,
                                    decay=0.9)

elif optimizer == "adam":
    opt_op =tf.train.AdamOptimizer(learning_rate=learning_rate,
                                beta1=0.9,
                                beta2=0.999)

else:
    raise NotImplementedError(value_error_warning_optimizer)

train_op =opt_op.minimize(loss=loss, global_step=global_step, name="train_op")

# -----Provide an estimator spec for `ModeKeys.TRAIN` mode-----
if (mode ==tf.estimator.ModeKeys.TRAIN):
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

def main(unused_argv):
    # -----Declare all hyperparameters of terminal interface-----
    phase =FLAGS.phase
    perform_valid_during_train =FLAGS.perform_valid_during_train # (optional) Note: if use validation dataset
                                                                # during train phase, the valid set and train set are
                                                                # in same data_dir
    log_step_count_steps =FLAGS.log_step_count_steps # (optional)
    save_checkpoints_steps =FLAGS.save_checkpoints_steps # (optional)
    keep_checkpoint_max =FLAGS.keep_checkpoint_max # (optional)
    model_dir =FLAGS.model_dir
    export_dir =FLAGS.export_dir
    data_dir =FLAGS.data_dir

```

```

delimiter =FLAGS.delimiter
separator =FLAGS.separator
batch_size =FLAGS.batch_size
epochs =FLAGS.epochs
shuffle =FLAGS.shuffle
buffer_size =FLAGS.buffer_size # (optional)
num_parallel_calls =FLAGS.num_parallel_calls # (optional)
use_dtype_high_precision =FLAGS.use_dtype_high_precision # (optional)
name_feat_inds =FLAGS.name_feat_inds # (optional)
name_feat_vals =FLAGS.name_feat_vals # (optional)
task =FLAGS.task
field_size =FLAGS.field_size
feat_size =FLAGS.feat_size
embed_size =FLAGS.embed_size
use_global_bias =FLAGS.use_global_bias # (optional)
use_linear =FLAGS.use_linear # (optional)
lamb =FLAGS.lamb
optimizer =FLAGS.optimizer
learning_rate =FLAGS.learning_rate

PREFIX_TRAIN_FILE ="train"
PREFIX_EVAL_FILE ="eval"
PREFIX_PREDICT_FILE ="predict"
REUSE =False
SEED =None

if use_dtype_high_precision ==False:
    dtype =tf.float32
else:
    dtype =tf.float64

hparams ={
    "task": task,
    "field_size": field_size,
    "feat_size": feat_size,
    "embed_size": embed_size,
    "use_global_bias": use_global_bias,
    "use_linear": use_linear,
    "lamb": lamb,
    "optimizer": optimizer,
    "learning_rate": learning_rate,
    "dtype": dtype,
    "name_feat_inds": name_feat_inds,
    "name_feat_vals": name_feat_vals,
    "reuse": REUSE,
    "seed": SEED
}
config =tf.estimator.RunConfig(
    log_step_count_steps=log_step_count_steps,
    save_checkpoints_steps=save_checkpoints_steps,
    keep_checkpoint_max=keep_checkpoint_max
)

```

```

estimator =tf.estimator.Estimator(model_fn=model_fn, model_dir=model_dir, params=hparams, config=config)

if phase == "train":
    filenames_train =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_TRAIN_FILE + "*"))
    if perform_valid_during_train ==True:
        filenames_valid =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_EVAL_FILE + "*"))

    train_spec =tf.estimator.TrainSpec(input_fn=lambda :input_fn(filenames=filenames_train,
                                                                delimiter=delimiter,
                                                                separator=separator,
                                                                batch_size=batch_size,
                                                                epochs=epochs,
                                                                shuffle=shuffle,
                                                                buffer_size=buffer_size,
                                                                num_parallel_calls=num_parallel_calls,
                                                                dtype=dtype,
                                                                name_feat_inds=name_feat_inds,
                                                                name_feat_vals=name_feat_vals),
                                         max_steps=None)

    eval_spec =tf.estimator.EvalSpec(input_fn=lambda :input_fn(filenames=filenames_valid,
                                                               delimiter=delimiter,
                                                               separator=separator,
                                                               batch_size=batch_size,
                                                               epochs=1,
                                                               shuffle=False,
                                                               buffer_size=buffer_size,
                                                               num_parallel_calls=num_parallel_calls,
                                                               dtype=dtype,
                                                               name_feat_inds=name_feat_inds,
                                                               name_feat_vals=name_feat_vals),
                                         steps=None,
                                         start_delay_secs=120,
                                         throttle_secs=600)

    tf.estimator.train_and_evaluate(estimator=estimator, train_spec=train_spec, eval_spec=eval_spec)
else:
    estimator.train(input_fn=lambda :input_fn(filenames=filenames_train,
                                              delimiter=delimiter,
                                              separator=separator,
                                              batch_size=batch_size,
                                              epochs=epochs,
                                              shuffle=shuffle,
                                              buffer_size=buffer_size,
                                              num_parallel_calls=num_parallel_calls,
                                              dtype=dtype,
                                              name_feat_inds=name_feat_inds,
                                              name_feat_vals=name_feat_vals),
                    steps=None,
                    max_steps=None)

elif phase == "eval":
    filenames_eval =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_EVAL_FILE + "*"))

    estimator.evaluate(input_fn=lambda :input_fn(filenames=filenames_eval,
                                                 delimiter=delimiter,
                                                 separator=separator,
                                                 batch_size=batch_size,

```

```

        epochs=1,
        shuffle=False,
        buffer_size=buffer_size,
        num_parallel_calls=num_parallel_calls,
        dtype=dtype,
        name_feat_inds=name_feat_inds,
        name_feat_vals=name_feat_vals))

elif phase == "predict":
    filenames_predict = tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_PREDICT_FILE + "*"))
    p = estimator.predict(input_fn=lambda : input_fn(filenames=filenames_predict,
                                                    delimiter=delimiter,
                                                    separator=separator,
                                                    batch_size=batch_size,
                                                    epochs=1,
                                                    shuffle=False,
                                                    buffer_size=buffer_size,
                                                    num_parallel_calls=num_parallel_calls,
                                                    dtype=dtype,
                                                    name_feat_inds=name_feat_inds,
                                                    name_feat_vals=name_feat_vals))

    # -----Usage demo, still need to be accomplished-----
    for ele in p:
        print(ele)

elif phase == "export":
    features ={
        name_feat_inds: tf.placeholder(dtype=tf.int32, shape=[None, field_size], name=name_feat_inds),
        name_feat_vals: tf.placeholder(dtype=dtype, shape=[None, field_size], name=name_feat_vals)
    }
    serving_input_receiver_fn =tf.estimator.export.build_raw_serving_input_receiver_fn(features=features)
    estimator.export_savedmodel(export_dir_base=export_dir,
                               serving_input_receiver_fn=serving_input_receiver_fn)
else:
    raise NotImplementedError("Argument <phase> value: {} is not supported.".format(phase))

if __name__ == '__main__':
    tf.logging.set_verbosity(v=tf.logging.INFO)
    tf.app.run(main=main)

```

54.2.4 References

- Field-aware Factorization Machines for CTR Prediction

54.3 Bilinear-Field-aware Factorization Machines (Bi-FFM)

54.3.1 Introduction

本节来自模型：Bilinear Field-aware Factorization Machines

Bilinear-FFM 的引入就是为了减少 FFM 模型的参数数量 其中对于参数 \mathbf{W} 的设置级别，有三种选择：

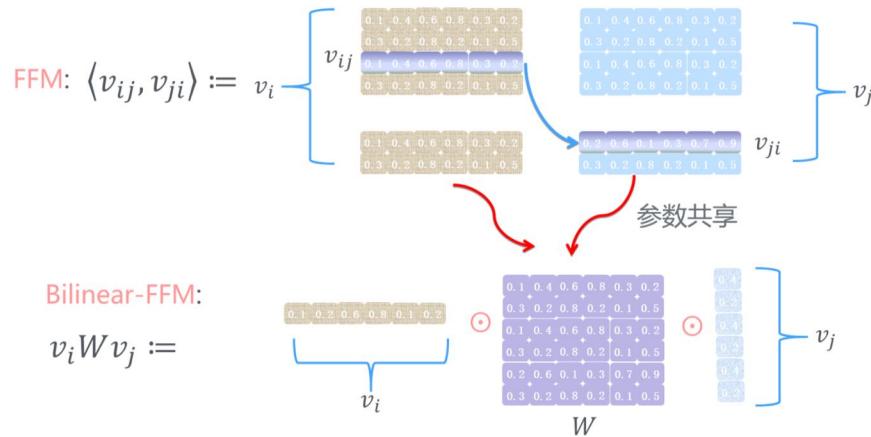


图 171: Basic idea of Bilinear-FFM model

- 所有特征 feature 共享参数矩阵 $\mathbf{W} \in \mathbb{R}^{K \times K}$ ，总参数数量相对 FM 增加 $K \times K$
- 一个特征域 (field) 一个 $\mathbf{W}_i \in \mathbb{R}^{K \times K}$ ，总参数数量相对 FM 增加 $F \times K \times K$
- 一个 field 组合一个 $\mathbf{W}_{ij} \in \mathbb{R}^{K \times K}$ ，总参数数量相对 FM 增加 $F \times F \times K \times K$

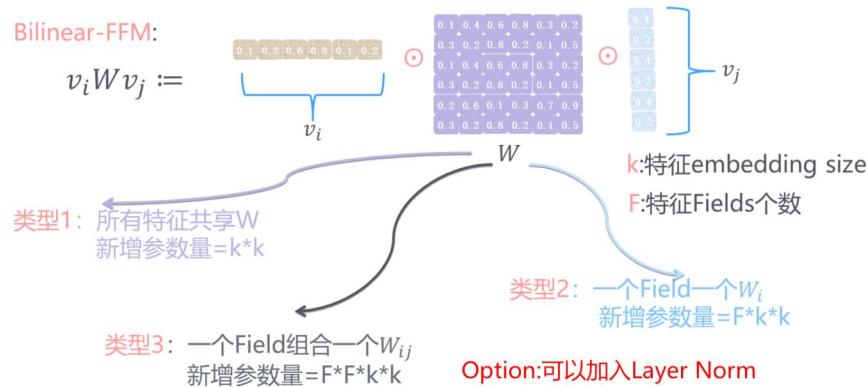


图 172: Three choices of parameter \mathbf{W}

54.3.2 Model formulation (未完成)

Model inference 以第三种选择为例，考虑 i -th field 和 j -th field 交叉的细节，则：

- $\because \mathbf{v}_i \in \mathbb{R}^{M \times K}$
- $\because \mathbf{v}_j \in \mathbb{R}^{K \times M}$
- $\therefore \mathbf{W}_{ij} \in \mathbb{R}^{K \times K}$
- $\therefore \mathbf{v}_i \odot \mathbf{W}_{ij} \odot \mathbf{v}_j \in \mathbb{R}$???

因此结果是：

- 结果优于或接近于 FFM 模型

Model Name	Criteo		Avazu	
	AUC	Logloss	AUC	Logloss
LR	0.7808	0.4681	0.7633	0.3891
FM	0.7923	0.4584	0.7745	0.3832
FFM	0.8001	0.4525	0.7795	0.3810
Bi-FFM-ALL	0.7935	0.4573	0.7754	0.3830
Bi-FFM-EACH	0.7963	0.4550	0.7738	0.3838
Bi-FFM-INTER	0.7995	0.4525	0.7781	0.3820
Bi-FFM-ALL-LN	0.8004	0.4518	0.7763	0.3837
Bi-FFM-EACH-LN	0.8007	0.4511	0.7745	0.3843
Bi-FFM-INTER-LN	0.8035	0.4484	0.7765	0.3829

Criteo和Avazu是两个工业级的CTR数据

一些结论：
1.FM显著好于LR；
2.FFM好于FM；
3.双线性FFM可以在大量减少参数情况下效果接近于FFM；
4.通常情况下，LN有助于提升效果

图 173: Result comparsion between bilinear-ffm models and other models

- 参数量为 FFM 模型的 2.6%

以Criteo数据集合为例子 (4500万数据) :

特征数量 : 230万 特征Field数量 : 39 假设特征embedding大小 : 10

FFM参数量 :	双线性FFM参数量 :
230万*39*10=8.97亿	FM : 230万*10=2300万
	类型1: 新增10*10=100
	类型2 : 新增39*10*10=3900
相差38倍 !	类型3 : 新增39*39*10*10=15万

图 174: Number of parameters comparsion between FFM models and bilinear-FFM model

54.4 Wide & Deep Network (WDN)

54.4.1 Introduction

本节来自论文：Wide & Deep Learning for Recommender Systems[?]

54.4.2 Model formulation

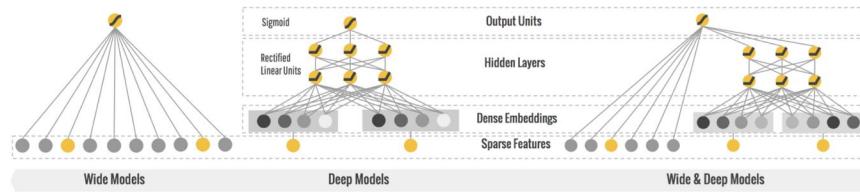


图 175: The structure of Wide & Deep Model[?]

Model inference

54.4.3 Model implement

代码如下：

```
"""
Created on 2019/03/22 16:35
author: Tong Jia
email: cecilio.jia@gmail.com
description:
    An implementation of DeepFM: A Factorization-Machine based Neural Network for CTR Prediction (DeepFM).
    See algorithm and hyperparameter details:
        [Guo et al., 2017] (https://www.ijcai.org/proceedings/2017/0239.pdf)
    The algorithm is developed with TensorFlow Estimator based on TensorFlow 1.12.0 version.
"""

import tensorflow as tf
from tensorflow.contrib.layers import xavier_initializer, l2_regularizer
import os

FLAGS = tf.flags.FLAGS
# -----Hyperparameters of estimator-----
tf.flags.DEFINE_enum(name="phase",
                     default=None,
                     enum_values=["train", "eval", "predict", "export"],
                     help="A string, representing the phase of estimator")
tf.flags.DEFINE_string(name="model_dir",
                      default=None,
                      help="A string, representing the folder path of saved model files")
tf.flags.DEFINE_string(name="export_dir",
                      default=None,
                      help="A string, representing the basic folder path of export .pb files")
# -----Hyperparameters of estimator (optional)-----
```

```

tf.flags.DEFINE_boolean(name="perform_valid_during_train",
                       default=True,
                       help="(optional) A boolean, instructing whether to perform validation on valid dataset
                             during train phase")

tf.flags.DEFINE_integer(name="log_step_count_steps",
                       default=500,
                       help="(optional) An integer, representing the frequency, in number of global steps, that
                             the global step/sec will be
                             logged during training",
                       lower_bound=1,
                       upper_bound=None)

tf.flags.DEFINE_integer(name="save_checkpoints_steps",
                       default=20000,
                       help="(optional) An integer, representing save checkpoints every this many steps",
                       lower_bound=1,
                       upper_bound=None)

tf.flags.DEFINE_integer(name="keep_checkpoint_max",
                       default=2,
                       help="(optional) An integer, representing max number of saved model checkpoints",
                       lower_bound=1,
                       upper_bound=None)

# -----Hyperparameters of input function-----
tf.flags.DEFINE_string(name="data_dir",
                       default=None,
                       help="A string, representing the folder path of dataset")

tf.flags.DEFINE_string(name="delimiter",
                       default=None,
                       help="A string, separating consecutive <index j>:<value j> pairs in a line of data file")

tf.flags.DEFINE_string(name="separator",
                       default=None,
                       help="A string, separating feature index(left part) and feature value(right part) in a
                             specific pair")

tf.flags.DEFINE_integer(name="batch_size",
                       default=None,
                       help="An integer, representing the number of consecutive elements of this dataset to
                             combine in a single batch",
                       lower_bound=1,
                       upper_bound=None)

tf.flags.DEFINE_integer(name="epochs",
                       default=None,
                       help="An integer, representing the number of times the dataset should be repeated",
                       lower_bound=1,
                       upper_bound=None)

# -----Hyperparameters of input function (optional)-----
tf.flags.DEFINE_boolean(name="shuffle",
                       default=True,
                       help="(optional) A boolean, instructing whether to randomly shuffle the samples of
                             training dataset")

tf.flags.DEFINE_integer(name="buffer_size",
                       default=100000,
                       help="(optional) An integer scalar, denoting the number of bytes to buffer",
                       lower_bound=1,

```

```

        upper_bound=None)
tf.flags.DEFINE_integer(name="num_parallel_calls",
                      default=4,
                      help="(optional) An integer scalar, representing the number elements to process in
                           parallel",
                      lower_bound=1,
                      upper_bound=None)
tf.flags.DEFINE_boolean(name="use_dtype_high_precision",
                      default=False,
                      help="(optional) A boolean, instructing the dtype of both input function and model
                           function. If False, use
                           tf.float32; if True, use
                           tf.float64")

tf.flags.DEFINE_string(name="name_feat_inds",
                      default="inds",
                      help="(optional) A string, representing the name of feature indices in return dict")
tf.flags.DEFINE_string(name="name_feat_vals",
                      default="vals",
                      help="(optional) A string, representing the name of feature values in return dict")
# -----Hyperparameters of model function-----
tf.flags.DEFINE_string(name="task",
                      default=None,
                      help="A string, representing the type of task (binary or regression)")
tf.flags.DEFINE_integer(name="field_size",
                      default=None,
                      help="An integer scalar, representing the number of fields(number of columns before
                           one-hot encoding) of dataset")
tf.flags.DEFINE_integer(name="feat_size",
                      default=None,
                      help="An integer scalar, representing the number of features(number of columns after
                           one-hot encoding) of dataset")
tf.flags.DEFINE_integer(name="embed_size",
                      default=None,
                      help="An integer scalar, representing the dimension of embedding vectors for all
                           features")
tf.flags.DEFINE_list(name="hidden_sizes",
                      default=None,
                      help="A list, containing the number of hidden units of each hidden layer in dnn part")
tf.flags.DEFINE_list(name="dropouts",
                      default=None,
                      help="A list or None, containing the dropout rate of each hidden layer in dnn part "
                           "(e.g. 0.4,0.3,0.2,0.1); if None, don't use dropout operation for any hidden layer")
# -----Hyperparameters of model function (optional)-----
tf.flags.DEFINE_boolean(name="use_global_bias",
                      default=True,
                      help="(optional) A boolean, instructing whether to use global bias in model inference")
tf.flags.DEFINE_boolean(name="use_linear",
                      default=True,
                      help="(optional) A boolean, instructing whether to use linear part in model inference")
tf.flags.DEFINE_boolean(name="use_hidden_bias",
                      default=True,
                      help="(optional) A boolean, instructing whether to use bias of hidden layer units in
                           parallel"
)

```

```

model inference")

tf.flags.DEFINE_boolean(name="use_bn",
                       default=True,
                       help="(optional) A boolean, instructing whether to use batch normalization for each
                             hidden layer in dnn part")

tf.flags.DEFINE_float(name="lamb",
                      default=0.001,
                      help="(optional) A float scalar, representing the coefficient of regularization term",
                      lower_bound=0.0,
                      upper_bound=None)

tf.flags.DEFINE_string(name="optimizer",
                      default="adam",
                      help="(optional) A string, representing the type of optimizer")

tf.flags.DEFINE_float(name="learning_rate",
                      default=0.003,
                      help="(optional) A float scalar, representing the learning rate of optimizer",
                      lower_bound=0.0,
                      upper_bound=None)

def input_fn(filenames,
             delimiter,
             separator,
             batch_size,
             epochs,
             shuffle=True,
             buffer_size=100000,
             num_parallel_calls=4,
             dtype=tf.float32,
             name_feat_inds="inds",
             name_feat_vals="vals"):

    """
    The input function for loading multi-field sparse dataset. The columns are
    separated by argument delimiter(e.g. " ") with the following schema (libsvm format):
    <label> <index 1>:<value 1> ... <index j>:<value j> ... <index d>:<value d> (d is the field size of dataset)
    e.g.
        0 0:0.732349 2:1 41:1 59:1 66:1 77:1 85:1 88:1 89:1 92:1 93:1 98:1 106:1
    where:
        delimiter is always " " in txt file format, "," in csv file format;
        separator is always equal to ":".
    Note:
        1. The input function is only used for dataset with one-hot active value in each field.
        2. In each line, the order of all fields must be fixed.
        3. The input function can be used for binary classification and regression task:
            binary classification: <label> in {0, 1};
            regression: <label> in (-inf, inf).
    Parameters
    -----
    :param filenames: list
        A list of string, containing one or more paths of filenames.
    :param delimiter: str
        A str, separating consecutive <index j>:<value j> pairs in data files.
    
```

```

:param separator: str
    A str, separating feature index(left part of key-value pair) and feature value(right part of key-value
    pair) in one specific pair.

:param batch_size: int
    An integer scalar, representing the number of consecutive elements of this dataset to combine in a
    single batch.

:param epochs: int
    An integer scalar, representing the number of times the dataset should be repeated.

:param shuffle: bool, optional
    A boolean(defaults to True), instructing whether to randomly shuffle the elements of this dataset.

:param buffer_size: int, optional
    An integer scalar(defaults to 100000), denoting the number of bytes to buffer.

:param num_parallel_calls: int, optional
    An integer scalar(defaults to 4), representing the number elements to process in parallel.

:param dtype: tf.Dtype, optional
    A tf.DType(defaults to tf.float32), representing the numeric type of values. it always takes value from
    [tf.float32, tf.float64].

:param name_feat_inds: str, optional
    A string, representing the name of feature indices in return dict.

:param name_feat_vals: str, optional
    A string, representing the name of feature values in return dict.

>Returns
-----
:return: dict
    A dict of two Tensors, representing features(including feature indices and feature values) in a single
    batch.

{
    <name_feat_inds>: tf.Tensor of feature indices in shape of (None, field_size),
    <name_feat_vals>: tf.Tensor of feature values in shape of (None, field_size)
}

:return: Tensor
    A Tensor in shape of (None), representing labels in a single batch.

"""

def map_func(line):
    columns = tf.string_split(source=[line], delimiter=delimiter, skip_empty=False).values
    label = tf.string_to_number(string_tensor=columns[0], out_type=dtype)
    splits = tf.string_split(source=columns[1:], delimiter=separator, skip_empty=False)
    feats = tf.reshape(
        tensor=splits.values, shape=splits.dense_shape
    ) # A tensor in shape of (field_size, 2), the first column contains feature indices, the second column
        # contains feature values
    inds, vals = tf.split(value=feats, num_or_size_splits=2, axis=1) # Two tensors in shape of (field_size,
        1)
    inds = tf.reshape(tensor=tf.string_to_number(string_tensor=inds, out_type=tf.int32), shape=[-1]) # A
        # tensor in shape of (field_size)
    vals = tf.reshape(tensor=tf.string_to_number(string_tensor=vals, out_type=dtype), shape=[-1]) # A
        # tensor in shape of (field_size)
    return {name_feat_inds: inds, name_feat_vals: vals}, label

dataset = tf.data.TextLineDataset(filenames=filenames, buffer_size=buffer_size). \
    map(map_func=map_func, num_parallel_calls=num_parallel_calls)

```

```

if (shuffle ==True):
    dataset =dataset.shuffle(buffer_size=buffer_size)
dataset =dataset. \
    repeat(count=epochs). \
    batch(batch_size=batch_size, drop_remainder=False)
dataset =dataset.prefetch(buffer_size=1)
iterator =dataset.make_one_shot_iterator()
batch_feats, batch_labels =iterator.get_next()
return batch_feats, batch_labels

def model_fn(features, labels, mode, params):
    """Model function of DeepFM for predictive analytics of high dimensional sparse data.

    Args of dict params:
        task: str
            A string, representing the type of task.
            Note:
                it must take value from ["binary", "regression"];
                it instruct the type of loss function:
                    "binary": sigmoid cross-entropy;
                    "regression": mean squared error.
        field_size: int
            An integer scalar, representing the number of fields(number of columns before one-hot encoding) of
            dataset.
        feat_size: int
            An integer scalar, representing the number of features(number of columns after one-hot encoding) of
            dataset.
        embed_size: int
            An integer scalar, representing the dimension of embedding vectors for all features.
        hidden_sizes: list
            A list, containing the number of hidden units of each hidden layer in dnn part.
            Note:
                it doesn't contain output layer of dnn part.
        dropouts: list or None
            If list, containing the dropout rate of each hidden layer in dnn part;
            If None, don't use dropout operation for any hidden layer.
            Note:
                if list, the length of <dropouts> must be equal to <hidden_sizes>.
        use_global_bias: bool
            A boolean, instructing whether to use global bias in model inference.
        use_linear: bool
            A boolean, instructing whether to use linear part in model inference.
        use_hidden_bias: bool
            A boolean, instructing whether to use bias of hidden layer units in model inference.
        use_bn: bool
            A boolean, instructing whether to use batch normalization for each hidden layer in dnn part.
        lamb: float
            A float scalar, representing the coefficient of regularization term (the larger the value of lamb,
            the stronger the penalty is).
        optimizer: str
            A string, representing the type of optimizer.

```

```

learning_rate: float
    A float scalar, representing the learning rate of optimizer.
dtype: tf.Dtype
    A tf.DType, representing the numeric type of values.
Note:
    it must take value from [tf.float32, tf.float64];
    it must be consistent with <dtype> of input function.
name_feat_inds: str
    A string, representing the name of feature indices in return dict.
Note:
    it must be consistent with <name_feat_inds> of input function.
name_feat_vals: str
    A string, representing the name of feature values in return dict.
Note:
    it must be consistent with <name_feat_vals> of input function.
reuse: bool
    A boolean, which takes value from [False, True, tf.AUTO_REUSE].
seed: int or None
    If integer scalar, representing the random seed of tensorflow;
    If None, random choice.
"""
# -----Declare all hyperparameters from params-----
task =params["task"]
field_size =params["field_size"]
feat_size =params["feat_size"]
embed_size =params["embed_size"]
hidden_sizes =params["hidden_sizes"]
dropouts =params["dropouts"]
use_global_bias =params["use_global_bias"]
use_linear =params["use_linear"]
use_hidden_bias =params["use_hidden_bias"]
use_bn =params["use_bn"]
lamb =params["lamb"]
optimizer =params["optimizer"]
learning_rate =params["learning_rate"]
dtype =params["dtype"]
name_feat_inds =params["name_feat_inds"]
name_feat_vals =params["name_feat_vals"]
reuse =params["reuse"]
seed =params["seed"]
# ----Hyperparameters for threshold for binary classification task
threshold =0.5
# -----Hyperparameters for exponential decay(*manual optional*)-----
decay_steps =5000
decay_rate =0.998
staircase =True
# ----Hyperparameters for information showing-----
name_probability_output ="prob"
name_classification_output ="class"
name_regression_output ="pred"
value_error_warning_task ="Argument of model function <task>: \"{}\" is invalid. It must be in
                                [\"binary\", \"regression\"]".format(task)

```

```

value_error_warning_optimizer ="Argument value of <optimizer>: {} is not supported.".format(optimizer)

if seed !=None:
    tf.set_random_seed(seed=seed)

# -----Assert for hyperparameters-----
assert (task in ["binary", "regression"])
if dropouts !=None:
    assert (len(dropouts) ==len(hidden_sizes))
assert (dtype in [tf.float32, tf.float64])

# -----Build model inference-----
with tf.variable_scope(name_or_scope="inference", reuse=reuse):
    with tf.name_scope(name="inputs"):
        ids =features[name_feat_inds]
        x =features[name_feat_vals]

    with tf.name_scope(name="global-bias"):
        b =tf.get_variable(name="b",
                           shape=[1],
                           dtype=dtype,
                           initializer=tf.zeros_initializer(dtype=dtype),
                           regularizer=l2_regularizer(scale=lamb),
                           trainable=use_global_bias)

    with tf.name_scope(name="lr-part"):
        W =tf.get_variable(name="W",
                           shape=[feat_size],
                           dtype=dtype,
                           initializer=xavier_initializer(uniform=True, dtype=dtype), # *manual optional*
                           regularizer=l2_regularizer(scale=lamb),
                           trainable=use_linear)
        # -----embedding lookup op for first order weights-----
        w =tf.nn.embedding_lookup(params=W, ids=ids) # A tensor in shape of (None, field_size)
        ylr =tf.reduce_sum(input_tensor=tf.multiply(x=x, y=w), # A tensor in shape of (None, field_size)
                           axis=1,
                           keepdims=False) # A tensor in shape of (None)

    with tf.name_scope(name="fm-part"):
        V =tf.get_variable(name="V",
                           shape=[feat_size, embed_size],
                           dtype=dtype,
                           initializer=xavier_initializer(uniform=False, dtype=dtype), # manual optional
                           regularizer=None, # *manual optional*
                           trainable=True)
        # -----embedding lookup op for second order weights-----
        v =tf.nn.embedding_lookup(params=V, ids=ids) # A tensor in shape of (None, field_size, embed_size)
        xreshape =tf.expand_dims(input=x, axis=-1) # A tensor in shape of (None, field_size, 1)
        vx =tf.multiply(x=xreshape, y=v) # A tensor in shape of (None, field_size, embed_size)
        p =tf.square(x=tx.reduce_sum(input_tensor=vx, axis=1, keepdims=False)) # a tensor in shape of
            # (None, embedding_size)
        q =tf.reduce_sum(input_tensor=tx.square(x=vx), # A tensor in shape of (None, field_size,

```

```

        embedding_size)
axis=1,
keepdims=False) # A tensor in shape of (None, embedding_size)
yfm = 0.5 *tf.reduce_sum(input_tensor=tf.subtract(x=p, y=q), # A tensor in shape of (None,
                           embedding_size)
axis=1,
keepdims=False) # A tensor in shape of (None)

with tf.name_scope(name="mlp-part"):
    ymlp =tf.reshape(tensor=vx, shape=[-1, field_size *embed_size]) # a tensor in shape of (None,
                                                                     field_size * embed_size)
    for l in range(len(hidden_sizes)):
        # -----The order for each hidden layer is: matmul => bn => relu => dropout => matmul => ...
        ymlp =tf.layers.dense(inputs=ymlp,
                              units=hidden_sizes[l],
                              activation=None,
                              use_bias=use_hidden_bias,
                              kernel_initializer=xavier_initializer(uniform=True, dtype=dtype),
                              bias_initializer=tf.zeros_initializer(dtype=dtype),
                              kernel_regularizer=None, # *manual optional*
                              bias_regularizer=None, # *manual optional*
                              trainable=True,
                              name="mlp-dense-hidden-{}".format(l))

        if use_bn ==True:
            ymlp =tf.layers.batch_normalization(inputs=ymlp,
                                                axis=-1,
                                                momentum=0.99, # *manual optional*
                                                epsilon=1e-3, # *manual optional*
                                                center=True,
                                                scale=True,
                                                beta_initializer=tf.zeros_initializer(dtype=dtype),
                                                gamma_initializer=tf.ones_initializer(dtype=dtype),
                                                moving_mean_initializer=tf.zeros_initializer(dtype=dtype),
                                                moving_variance_initializer=tf.ones_initializer(dtype=dtype),
                                                beta_regularizer=None, # *manual optional*
                                                gamma_regularizer=None, # *manual optional*
                                                training=(mode ==tf.estimator.ModeKeys.TRAIN),
                                                trainable=True,
                                                name="mlp-bn-hidden-{}".format(l))

        ymlp =tf.nn.relu(features=ymlp)
        if dropouts !=None:
            ymlp =tf.layers.dropout(inputs=ymlp,
                                   rate=dropouts[l],
                                   seed=seed,
                                   training=(mode ==tf.estimator.ModeKeys.TRAIN))

# -----output layer of mlp part-----
ymlp =tf.layers.dense(inputs=ymlp,
                      units=1,
                      activation=None,
                      use_bias=False,
                      kernel_initializer=xavier_initializer(uniform=True, dtype=dtype),

```

```

        kernel_regularizer=None, # *manual optional*
        trainable=True,
        name="mlp-dense-output") # A tensor in shape of (None, 1)
ymlp =tf.squeeze(input=ymlp, axis=1) # A tensor in shape of (None)

with tf.name_scope(name="output"):
    if use_global_bias ==False and use_linear ==False:
        logits =yfm +ymlp
    elif use_global_bias ==False and use_linear ==True:
        logits =ylr +yfm +ymlp
    elif use_global_bias ==True and use_linear ==False:
        logits =b +yfm +ymlp
    else:
        logits =b +ylr +yfm +ymlp

    if task =="binary":
        logits =tf.squeeze(input=logits, axis=1) # A tensor in shape of (None)
        probs =tf.nn.sigmoid(x=logits)
        classes =tf.cast(x=tf.greater(x=tf.nn.sigmoid(x=logits), y=threshold), dtype=tf.int32)
        predictions =[{
            name_probability_output: probs,
            name_classification_output: classes
        }]
    elif task =="regression":
        logits =tf.squeeze(input=logits, axis=1) # A tensor in shape of (None)
        predictions =[{
            name_regression_output: logits
        }]
    else:
        raise ValueError(value_error_warning_task)

# -----Provide an estimator spec for `ModeKeys.PREDICTION` mode-----
if mode ==tf.estimator.ModeKeys.PREDICT:
    export_outputs =[{
        tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
            tf.estimator.export.PredictOutput(outputs=predictions)
    } # For usage of tensorflow serving
    return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions, export_outputs=export_outputs)

# -----Build loss function-----
if task =="binary":
    loss =tf.reduce_mean(input_tensor=tf.nn.sigmoid_cross_entropy_with_logits(labels=labels,
                                                                           logits=logits),
                           axis=0,
                           keepdims=False) # A scalar, representing the training loss of current batch training
                           dataset
elif task =="regression":
    loss =tf.reduce_mean(input_tensor=tf.square(x=tf.subtract(x=labels, y=logits)),
                           axis=0,
                           keepdims=False) # A scalar, representing the training loss of current batch training
                           dataset
else:

```

```

        raise ValueError(value_error_warning_task)

reg = tf.reduce_sum(input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
                     axis=0,
                     keepdims=False,
                     name="regularization") # A scalar, representing the regularization loss of current batch
                                         training dataset
loss += reg

# -----Provide an estimator spec for `ModeKeys.EVAL` mode-----
if mode == tf.estimator.ModeKeys.EVAL:
    if task == "binary":
        eval_metric_ops = {
            "accuracy": tf.metrics.accuracy(labels=labels,
                                              predictions=predictions[name_classification_output]),
            "precision": tf.metrics.precision(labels=labels,
                                              predictions=predictions[name_classification_output]),
            "recall": tf.metrics.recall(labels=labels, predictions=predictions[name_classification_output]),
            "auc": tf.metrics.auc(labels=labels, predictions=predictions[name_classification_output])
        }
    elif task == "regression":
        eval_metric_ops = {
            "rmse": tf.metrics.root_mean_squared_error(labels=labels,
                                                       predictions=predictions[name_regression_output])
        }
    else:
        raise ValueError(value_error_warning_task)
return tf.estimator.EstimatorSpec(mode=mode, loss=loss, predictions=predictions,
                                   eval_metric_ops=eval_metric_ops)

# -----Build optimizer-----
global_step = tf.train.get_or_create_global_step(graph=tf.get_default_graph()) # Define a global step for
                                         training step counter
if optimizer == "sgd":
    opt_op = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
elif optimizer == "sgd-exp-decay":
    decay_learning_rate = tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)
    opt_op = tf.train.GradientDescentOptimizer(learning_rate=decay_learning_rate)
elif optimizer == "momentum":
    opt_op = tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                        momentum=0.9,
                                        use_nesterov=False)
elif optimizer == "momentum-exp-decay":
    decay_learning_rate = tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)

```

```

opt_op =tf.train.MomentumOptimizer(learning_rate=decay_learning_rate,
                                    momentum=0.9,
                                    use_nesterov=False)
elif optimizer == "nesterov":
    opt_op =tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                    momentum=0.9,
                                    use_nesterov=True)
elif optimizer == "nesterov-exp-decay":
    decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)
opt_op =tf.train.MomentumOptimizer(learning_rate=decay_learning_rate,
                                    momentum=0.9,
                                    use_nesterov=True)
elif optimizer == "adagrad":
    opt_op =tf.train.AdagradOptimizer(learning_rate=learning_rate,
                                    initial_accumulator_value=0.1)
elif optimizer == "adadelta":
    opt_op =tf.train.AdadeltaOptimizer(learning_rate=learning_rate,
                                    rho=0.95)
elif optimizer == "rmsprop":
    opt_op =tf.train.RMSPropOptimizer(learning_rate=learning_rate,
                                    decay=0.9)
elif optimizer == "adam":
    opt_op =tf.train.AdamOptimizer(learning_rate=learning_rate,
                                beta1=0.9,
                                beta2=0.999)
else:
    raise NotImplementedError(value_error_warning_optimizer)

train_op =opt_op.minimize(loss=loss, global_step=global_step, name="train_op")

# -----Provide an estimator spec for `ModeKeys.TRAIN` mode-----
if (mode ==tf.estimator.ModeKeys.TRAIN):
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

def main(unused_argv):
    # -----Declare all hyperparameters of terminal interface-----
    phase =FLAGS.phase
    perform_valid_during_train =FLAGS.perform_valid_during_train # (optional) Note: if use validation dataset
                                                                # during train phase, the valid set and train set are
                                                                # in same data_dir
    log_step_count_steps =FLAGS.log_step_count_steps # (optional)
    save_checkpoints_steps =FLAGS.save_checkpoints_steps # (optional)
    keep_checkpoint_max =FLAGS.keep_checkpoint_max # (optional)
    model_dir =FLAGS.model_dir
    export_dir =FLAGS.export_dir
    data_dir =FLAGS.data_dir
    delimiter =FLAGS.delimiter

```

```

separator =FLAGS.separator
batch_size =FLAGS.batch_size
epochs =FLAGS.epochs
shuffle =FLAGS.shuffle
buffer_size =FLAGS.buffer_size # (optional)
num_parallel_calls =FLAGS.num_parallel_calls # (optional)
use_dtype_high_precision =FLAGS.use_dtype_high_precision # (optional)
name_feat_inds =FLAGS.name_feat_inds # (optional)
name_feat_vals =FLAGS.name_feat_vals # (optional)
task =FLAGS.task
field_size =FLAGS.field_size
feat_size =FLAGS.feat_size
embed_size =FLAGS.embed_size
hidden_sizes =FLAGS.hidden_sizes
dropouts =FLAGS.dropouts
use_global_bias =FLAGS.use_global_bias # (optional)
use_linear =FLAGS.use_linear # (optional)
use_hidden_bias =FLAGS.use_hidden_bias # (optional)
use_bn =FLAGS.use_bn # (optional)
lamb =FLAGS.lamb
optimizer =FLAGS.optimizer
learning_rate =FLAGS.learning_rate

PREFIX_TRAIN_FILE ="train"
PREFIX_EVAL_FILE ="eval"
PREFIX_PREDICT_FILE ="predict"
REUSE =False
SEED =None

if use_dtype_high_precision ==False:
    dtype =tf.float32
else:
    dtype =tf.float64
hidden_sizes =[int(float(ele)) for ele in hidden_sizes]
if dropouts !=None:
    dropouts =[float(ele) for ele in dropouts]

hparams ={
    "task": task,
    "field_size": field_size,
    "feat_size": feat_size,
    "embed_size": embed_size,
    "hidden_sizes": hidden_sizes,
    "dropouts": dropouts,
    "use_global_bias": use_global_bias,
    "use_linear": use_linear,
    "use_hidden_bias": use_hidden_bias,
    "use_bn": use_bn,
    "lamb": lamb,
    "optimizer": optimizer,
    "learning_rate": learning_rate,
    "dtype": dtype,
}

```

```

    "name_feat_inds": name_feat_inds,
    "name_feat_vals": name_feat_vals,
    "reuse": REUSE,
    "seed": SEED
}

config =tf.estimator.RunConfig(
    log_step_count_steps=log_step_count_steps,
    save_checkpoints_steps=save_checkpoints_steps,
    keep_checkpoint_max=keep_checkpoint_max
)
estimator =tf.estimator.Estimator(model_fn=model_fn, model_dir=model_dir, params=hparams, config=config)
if phase == "train":
    filenames_train =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_TRAIN_FILE + "*"))
    if perform_valid_during_train ==True:
        filenames_valid =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_EVAL_FILE + "*"))
        train_spec =tf.estimator.TrainSpec(input_fn=lambda :input_fn(filenames=filenames_train,
                                                                     delimiter=delimiter,
                                                                     separator=separator,
                                                                     batch_size=batch_size,
                                                                     epochs=epochs,
                                                                     shuffle=shuffle,
                                                                     buffer_size=buffer_size,
                                                                     num_parallel_calls=num_parallel_calls,
                                                                     dtype=dtype,
                                                                     name_feat_inds=name_feat_inds,
                                                                     name_feat_vals=name_feat_vals),
                                             max_steps=None)
        eval_spec =tf.estimator.EvalSpec(input_fn=lambda :input_fn(filenames=filenames_valid,
                                                                    delimiter=delimiter,
                                                                    separator=separator,
                                                                    batch_size=batch_size,
                                                                    epochs=1,
                                                                    shuffle=False,
                                                                    buffer_size=buffer_size,
                                                                    num_parallel_calls=num_parallel_calls,
                                                                    dtype=dtype,
                                                                    name_feat_inds=name_feat_inds,
                                                                    name_feat_vals=name_feat_vals),
                                         steps=None,
                                         start_delay_secs=120,
                                         throttle_secs=600)
        tf.estimator.train_and_evaluate(estimator=estimator, train_spec=train_spec, eval_spec=eval_spec)
    else:
        estimator.train(input_fn=lambda :input_fn(filenames=filenames_train,
                                                   delimiter=delimiter,
                                                   separator=separator,
                                                   batch_size=batch_size,
                                                   epochs=epochs,
                                                   shuffle=shuffle,
                                                   buffer_size=buffer_size,
                                                   num_parallel_calls=num_parallel_calls,
                                                   dtype=dtype,

```

```

        name_feat_inds=name_feat_inds,
        name_feat_vals=name_feat_vals),
    steps=None,
    max_steps=None)
elif phase == "eval":
    filenames_eval =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_EVAL_FILE +"*"))
estimator.evaluate(input_fn=lambda :input_fn(filenames=filenames_eval,
                                             delimiter=delimiter,
                                             separator=separator,
                                             batch_size=batch_size,
                                             epochs=1,
                                             shuffle=False,
                                             buffer_size=buffer_size,
                                             num_parallel_calls=num_parallel_calls,
                                             dtype=dtype,
                                             name_feat_inds=name_feat_inds,
                                             name_feat_vals=name_feat_vals))

elif phase == "predict":
    filenames_predict =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_PREDICT_FILE +"*"))
    p = estimator.predict(input_fn=lambda :input_fn(filenames=filenames_predict,
                                                    delimiter=delimiter,
                                                    separator=separator,
                                                    batch_size=batch_size,
                                                    epochs=1,
                                                    shuffle=False,
                                                    buffer_size=buffer_size,
                                                    num_parallel_calls=num_parallel_calls,
                                                    dtype=dtype,
                                                    name_feat_inds=name_feat_inds,
                                                    name_feat_vals=name_feat_vals))

# -----Usage demo, still need to be accomplished-----
for ele in p:
    print(ele)
elif phase == "export":
    features ={
        name_feat_inds: tf.placeholder(dtype=tf.int32, shape=[None, field_size], name=name_feat_inds),
        name_feat_vals: tf.placeholder(dtype=dtype, shape=[None, field_size], name=name_feat_vals)
    }
    serving_input_receiver_fn =tf.estimator.export.build_raw_serving_input_receiver_fn(features=features)
    estimator.export_savedmodel(export_dir_base=export_dir,
                               serving_input_receiver_fn=serving_input_receiver_fn)
else:
    raise NotImplementedError("Argument <phase> value: {} is not supported.".format(phase))

if __name__ == '__main__':
    tf.logging.set_verbosity(v=tf.logging.INFO)
    tf.app.run(main=main)

```

54.5 Deep & Cross Network (DCN)

54.5.1 Introduction

本节来自论文：Deep & Cross Network for Ad Click Predictions[?]

54.5.2 Model formulation

Model inference 网络的结构如图所示：模型主要包含了两部分结构：

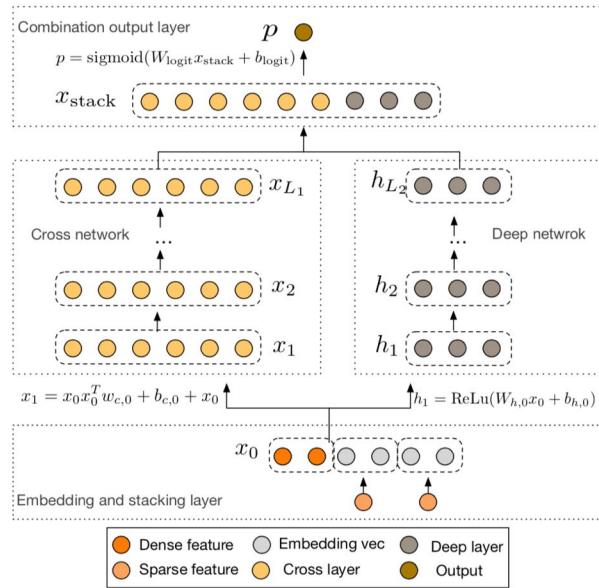


图 176: The Deep & Cross Network[?]

- Cross network
- Deep network

Cross network

$$\mathbf{x}_{l+1} = \mathbf{x}_0 \mathbf{x}_l^T \mathbf{w}_l + \mathbf{b}_l + \mathbf{x}_l = f(\mathbf{x}_l, \mathbf{w}_l, \mathbf{b}_l) + \mathbf{x}_l$$

展开进行剖析：

- $\mathbf{x}_0 \in \mathbb{R}^d$
- $\mathbf{x}_l \in \mathbb{R}^d, \quad \forall l \in \{1, \dots, L_1\}$
- $\mathbf{w}_l \in \mathbb{R}^d, \quad \forall l \in \{1, \dots, L_1\}$

$$\mathbf{x}_l^T \mathbf{w}_l = (x_{l,1}, \dots, x_{l,j}, \dots, x_{l,d}) \begin{bmatrix} w_{l,1} \\ \vdots \\ w_{l,j} \\ \vdots \\ w_{l,d} \end{bmatrix} = \sum_{j=1}^d w_{l,j} x_{l,j} \in \mathbb{R}$$

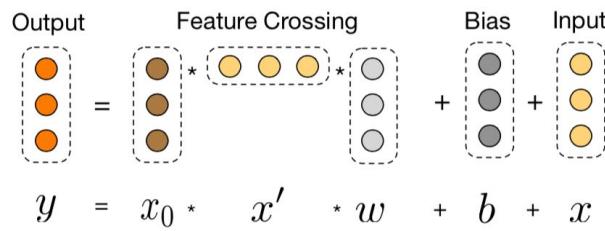


图 177: Visualization of a cross layer

54.5.3 Model implement

```

"""
Created on 2019/04/02 15:56
author: Tong Jia
email: cecilio.jia@gmail.com
description:
    An implementation of Deep & Cross Network for Ad Click Predictions (DCN).
    See algorithm and hyperparameter details:
        [Wang et al., 2017] (https://arxiv.org/pdf/1708.05123.pdf)
    The algorithm is developed with TensorFlow Estimator based on TensorFlow 1.12.0 version.
"""

import tensorflow as tf
from tensorflow.contrib.layers import xavier_initializer, l2_regularizer
import os

FLAGS = tf.flags.FLAGS
# -----Hyperparameters of estimator-----
tf.flags.DEFINE_enum(name="phase",
                     default=None,
                     enum_values=["train", "eval", "predict", "export"],
                     help="A string, representing the phase of estimator")
tf.flags.DEFINE_string(name="model_dir",
                      default=None,
                      help="A string, representing the folder path of saved model files")
tf.flags.DEFINE_string(name="export_dir",
                      default=None,
                      help="A string, representing the basic folder path of export .pb files")
# -----Hyperparameters of estimator (optional)-----
tf.flags.DEFINE_boolean(name="perform_valid_during_train",
                       default=True,
                       help="(optional) A boolean, instructing whether to perform validation on valid dataset
                             during train phase")
tf.flags.DEFINE_integer(name="log_step_count_steps",
                       default=500,
                       help="(optional) An integer, representing the frequency, in number of global steps, that
                             the global step/sec will be
                             logged during training",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="save_checkpoints_steps",
                       default=20000,
                       help="(optional) An integer, representing save checkpoints every this many steps",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="keep_checkpoint_max",
                       default=2,
                       help="(optional) An integer, representing max number of saved model checkpoints",
                       lower_bound=1,
                       upper_bound=None)
# -----Hyperparameters of input function-----
tf.flags.DEFINE_string(name="data_dir",

```

```

        default=None,
        help="A string, representing the folder path of dataset")
tf.flags.DEFINE_string(name="delimiter",
                      default=None,
                      help="A string, separating consecutive columns in a line of data file")
tf.flags.DEFINE_integer(name="field_size_numerical",
                       default=None,
                       help="An integer scalar, representing the number of numerical fields of dataset",
                       lower_bound=0,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="field_size_categorical",
                       default=None,
                       help="An integer scalar, representing the number of categorical fields of dataset",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="batch_size",
                       default=None,
                       help="An integer, representing the number of consecutive elements of this dataset to
                            combine in a single batch",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="epochs",
                       default=None,
                       help="An integer, representing the number of times the dataset should be repeated",
                       lower_bound=1,
                       upper_bound=None)
# -----Hyperparameters of input function (optional)-----
tf.flags.DEFINE_boolean(name="shuffle",
                       default=True,
                       help="(optional) A boolean, instructing whether to randomly shuffle the samples of
                            training dataset")
tf.flags.DEFINE_integer(name="buffer_size",
                       default=100000,
                       help="(optional) An integer scalar, denoting the number of bytes to buffer",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="num_parallel_calls",
                       default=4,
                       help="(optional) An integer scalar, representing the number elements to process in
                            parallel",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_boolean(name="use_dtype_high_precision",
                       default=False,
                       help="(optional) A boolean, instructing the dtype of both input function and model
                            function. "
                            "If False, use tf.float32; if True, use tf.float64")
tf.flags.DEFINE_string(name="name_feat_vals_numerical",
                      default="inds",
                      help="(optional) A string, representing the name of numerical feature values in return
                           dict")
tf.flags.DEFINE_string(name="name_feat_inds_categorical",

```

```

        default="vals",
        help="(optional) A string, representing the name of categorical feature indices in return
dict")

# -----Hyperparameters of model function-----
tf.flags.DEFINE_string(name="task",
                       default=None,
                       help="A string, representing the type of task (binary, multi or regression)")

tf.flags.DEFINE_integer(name="output_size",
                       default=None,
                       help="An integer scalar, representing the number of output units",
                       lower_bound=1,
                       upper_bound=None)

tf.flags.DEFINE_integer(name="feat_size_categorical",
                       default=None,
                       help="An integer scalar, representing the number of categorical features of dataset",
                       lower_bound=1,
                       upper_bound=None)

tf.flags.DEFINE_integer(name="embed_size",
                       default=None,
                       help="An integer scalar, representing the dimension of embedding vectors for all
features",
                       lower_bound=1,
                       upper_bound=None)

tf.flags.DEFINE_integer(name="num_cross_hidden_layers",
                       default=None,
                       help="An integer scalar, representing the number of hidden layers belongs to cross part",
                       lower_bound=1,
                       upper_bound=None)

tf.flags.DEFINE_list(name="deep_hidden_sizes",
                     default=None,
                     help="A list, containing the number of hidden units of each hidden layer in deep part (e.g.
128,64,32,16)")

tf.flags.DEFINE_list(name="dropouts",
                     default=None,
                     help="A list or None, containing the dropout rate of each hidden layer in dnn part (e.g.
0.4,0.3,0.2,0.1); "
                     "if None, don't use dropout operation for any hidden layer")

# -----Hyperparameters of model function (optional)-----
tf.flags.DEFINE_boolean(name="use_global_bias",
                       default=True,
                       help="(optional) A boolean, instructing whether to use global bias in model inference")

tf.flags.DEFINE_boolean(name="use_deep_hidden_bias",
                       default=True,
                       help="(optional) A boolean, instructing whether to use bias of hidden layer units in deep
part of model inference")

tf.flags.DEFINE_boolean(name="use_bn",
                       default=True,
                       help="(optional) A boolean, instructing whether to use batch normalization for each
hidden layer in dnn part")

tf.flags.DEFINE_float(name="lamb",
                      default=0.001,
                      help="(optional) A float scalar, representing the coefficient of regularization term",

```

```

        lower_bound=0.0,
        upper_bound=None)
tf.flags.DEFINE_string(name="optimizer",
                      default="adam",
                      help="(optional) A string, representing the type of optimizer")
tf.flags.DEFINE_float(name="learning_rate",
                      default=0.003,
                      help="(optional) A float scalar, representing the learning rate of optimizer",
                      lower_bound=0.0,
                      upper_bound=None)

def input_fn(filenames,
            delimiter,
            field_size_numerical,
            field_size_categorical,
            batch_size,
            epochs,
            shuffle=True,
            buffer_size=100000,
            num_parallel_calls=4,
            dtype=tf.float32,
            name_feat_vals_numerical="vals_numerical",
            name_feat_inds_categorical="inds_categorical"):
    """
The input function for loading sparse dataset in Deep & Cross format. The columns are separated by argument
delimiter(e.g. " ").
Note:
1. From left to right in each line, there contains three parts in order:
label => numerical fields => categorical fields
<label>
<value 1 numerical> ... <value j_n numerical>
<index 1 categorical> ... <index j_c categorical>
2. Categorical fields group maintains an index-system independently.
3. The feature type of each fields group:
numerical fields(fixed length >= 0): dense fields
categorical fields(fixed length > 1): each must be one-hot active field.
4. The order of numerical fields must be fixed, and the order of categorical fields must be fixed too.
5. The input function can be used for binary classification, multi classification and regression task:
binary classification: <label> in {0, 1};
multi classification: <label> in {0, K} (K is the number of total classes);
regression: <label> in (-inf, inf).
e.g.
(field_size_numerical = 2, field_size_categorical = 6, feat_size_categorical = 300)
0 0.172351 0.413592 1 14 20 134 231 293
1 0.314512 0.871236 4 17 78 104 280 298
For the first sample:
0:                  label
0.172351          value of 1-th numerical field
0.413592          value of 2-th numerical field
1 14 20 134 231 293 indices of categorical fields
    
```

```

Parameters
-----
:param filenames: list
    A list of string, containing one or more paths of filenames.
:param delimiter: str
    A str, separating consecutive columns in data files.
:param field_size_numerical: int
    An integer scalar, representing the number of numerical fields(namely the number of numerical features)
    of dataset.
:param field_size_categorical: int
    An integer scalar, representing the number of categorical fields(number of categorical columns before
    one-hot encoding) of dataset.
:param batch_size: int
    An integer scalar, representing the number of consecutive elements of this dataset to combine in a
    single batch.
:param epochs: int
    An integer scalar, representing the number of times the dataset should be repeated.
:param shuffle: bool, optional
    A boolean(defaults to True), instructing whether to randomly shuffle the elements of this dataset.
:param buffer_size: int, optional
    An integer scalar(defaults to 100000), denoting the number of bytes to buffer.
:param num_parallel_calls: int, optional
    An integer scalar(defaults to 4), representing the number elements to process in parallel.
:param dtype: tf.Dtype, optional
    A tf.DType(defaults to tf.float32), representing the numeric type of values. it always takes value from
    [tf.float32, tf.float64].
:param name_feat_vals_numerical: str, optional
    A string, representing the name of numerical feature values in return dict.
:param name_feat_inds_categorical: str, optional
    A string, representing the name of categorical feature indices in return dict.

Returns
-----
:return: dict
    A dict of two Tensors, representing features(including feature indices and feature values) in a single
    batch.
    {
        <name_feat_vals_numerical>: tf.Tensor of numerical feature values in shape of (None,
                                         field_size_numerical),
        <name_feat_inds_categorical>: tf.Tensor of categorical feature indices in shape of (None,
                                         field_size_categorical)
    }
:return: Tensor
    A Tensor in shape of (None), representing labels in a single batch.
"""

def map_func(line):
    columns = tf.string_split(source=[line], delimiter=delimiter, skip_empty=False).values
    # -----Process label-----
    label = tf.string_to_number(string_tensor=columns[0], out_type=dtype)
    # -----Process numerical fields-----
    vals_numerical = tf.string_to_number(string_tensor=columns[1: 1 + field_size_numerical],
                                         out_type=dtype) # A tensor in shape of (field_size_numerical)

```

```

# -----Process categorical fields-----
inds_categorical =tf.string_to_number(string_tensor=columns[-field_size_categorical: ],
                                       out_type=tf.int32) # A tensor in shape of (field_size_categorical)
feats =[

    name_feat_vals_numerical: vals_numerical,
    name_feat_inds_categorical: inds_categorical
]
return feats, label

dataset =tf.data.TextLineDataset(filenames=filenames, buffer_size=buffer_size).\
    map(map_func=map_func, num_parallel_calls=num_parallel_calls)
if shuffle:
    dataset =dataset.shuffle(buffer_size=buffer_size)
dataset =dataset. \
    repeat(count=epochs). \
    batch(batch_size=batch_size, drop_remainder=False)
dataset =dataset.prefetch(buffer_size=1)
iterator =dataset.make_one_shot_iterator()
batch_feats, batch_labels =iterator.get_next()
return batch_feats, batch_labels

def model_fn(features, labels, mode, params):
    """Model function of Deep & Cross network(DCN) for predictive analytics of high dimensional sparse data.

Args of dict params:
task: str
    A string, representing the type of task.
Note:
    it must take value from ["binary", "multi", "regression"];
    it instruct the type of loss function:
        "binary": sigmoid cross-entropy;
        "multi": softmax cross-entropy;
        "regression": mean squared error.
output_size: int
    An integer scalar, representing the number of output units.
Note:
    it must be correspond to <task>;
    task == "binary": output_size must be equal to 1;
    task == "multi": output_size must be equal to the dimension of class distribution;
    task == "regression": output_size must be equal to 1.
field_size_numerical: int
    An integer scalar, representing the number of numerical fields(also is the number of numerical
    features) of dataset.
Note:
    it must be consistent with <field_size_numerical> of input function.
field_size_categorical: int
    An integer scalar, representing the number of categorical fields(number of categorical columns
    before one-hot encoding) of dataset.
Note:
    it must be consistent with <field_size_categorical> of input function.
feat_size_categorical: int

```

```

    An integer scalar, representing the number of categorical features(number of categorical columns
    after one-hot encoding) of dataset.

embed_size: int
    An integer scalar, representing the dimension of embedding vectors for all categorical features.

num_cross_hidden_layers: int
    An integer scalar, representing the number of hidden layers belongs to cross part.

deep_hidden_sizes: list
    A list, containing the number of hidden units of each hidden layer in deep part.

Note:
    it doesn't contain output layer of dnn part.

dropouts: list or None
    If list, containing the dropout rate of each hidden layer in dnn part;
    If None, don't use dropout operation for any hidden layer.

Note:
    if list, the length of <dropouts> must be equal to <hidden_sizes>.

use_global_bias: bool
    A boolean, instructing whether to use global bias in output part of model inference.

use_deep_hidden_bias: bool
    A boolean, instructing whether to use bias of hidden layer units in deep part of model inference.

use_bn: bool
    A boolean, instructing whether to use batch normalization for each hidden layer in deep part.

lamb: float
    A float scalar, representing the coefficient of regularization term (the larger the value of lamb,
    the stronger the penalty is).

optimizer: str
    A string, representing the type of optimizer.

learning_rate: float
    A float scalar, representing the learning rate of optimizer.

dtype: tf.Dtype
    A tf.DType, representing the numeric type of values.

Note:
    it must take value from [tf.float32, tf.float64];
    it must be consistent with <dtype> of input function.

name_feat_vals_numerical: str, optional
    A string, representing the name of numerical feature values in return dict.

Note:
    it must be consistent with <name_feat_vals_numerical> of input function.

name_feat_inds_categorical: str, optional
    A string, representing the name of categorical feature indices in return dict.

Note:
    it must be consistent with <name_feat_inds_categorical> of input function.

reuse: bool
    A boolean, which takes value from [False, True, tf.AUTO_REUSE].

seed: int or None
    If integer scalar, representing the random seed of tensorflow;
    If None, random choice.

"""
# -----Declare all hyperparameters from params-----
task = params["task"]
output_size = params["output_size"]
field_size_numerical = params["field_size_numerical"]
field_size_categorical = params["field_size_categorical"]

```

```

feat_size_categorical =params["feat_size_categorical"]
embed_size =params["embed_size"]
num_cross_hidden_layers =params["num_cross_hidden_layers"]
deep_hidden_sizes =params["deep_hidden_sizes"]
dropouts =params["dropouts"]
use_global_bias =params["use_global_bias"]
use_deep_hidden_bias =params["use_deep_hidden_bias"]
use_bn =params["use_bn"]
lamb =params["lamb"]
optimizer =params["optimizer"]
learning_rate =params["learning_rate"]
dtype =params["dtype"]
name_feat_vals_numerical =params["name_feat_vals_numerical"]
name_feat_inds_categorical =params["name_feat_inds_categorical"]
reuse =params["reuse"]
seed =params["seed"]
# -----Hyperparameters for threshold for binary classification task
threshold =0.5
# -----Hyperparameters for exponential decay(*manual optional)-----
decay_steps =5000
decay_rate =0.998
staircase =True
# -----Hyperparameters for information showing-----
name_probability_output ="prob"
name_classification_output ="class"
name_regression_output ="pred"
value_error_warning_task ="Argument of model function <task>: \"{}\" is not supported. It must be in " \
                           "[\"binary\", \"multi\", \"regression\"]".format(task)
value_error_warning_optimizer ="Argument value of <optimizer>: {} is not supported.".format(optimizer)
value_error_warning_output_size_and_task ="Argument of model function <output_size>: {}, must be 1 when" \
                                         "<task> " \
                                         "is: \"{}\"".format(output_size, task)

# -----Assert for hyperparameters-----
if task == "binary":
    assert (output_size == 1), value_error_warning_output_size_and_task
if task == "regression":
    assert (output_size == 1), value_error_warning_output_size_and_task

if seed !=None:
    tf.set_random_seed(seed)

with tf.variable_scope(name_or_scope="inference", reuse=reuse):
    with tf.name_scope(name="inputs"):
        valsn =features[name_feat_vals_numerical]
        indsc =features[name_feat_inds_categorical] # A tensor in shape of (None, field_size_categorical)
        batch =tf.shape(input=valsn)[0]
        dim =field_size_numerical +field_size_categorical *embed_size

    with tf.name_scope(name="embed-and-stack-layer"):
        V =tf.get_variable(name="V",
                           shape=[feat_size_categorical, embed_size],

```

```

        dtype=dtype,
        initializer=xavier_initializer(uniform=False, seed=seed, dtype=dtype),
        regularizer=None)
embed =tf.nn.embedding_lookup(params=V, ids=indsc) # A tensor in shape of (None,
                                                field_size_categorical, embed_size)
embed =tf.reshape(tensor=embed, shape=[-1, field_size_categorical *embed_size])
x = tf.reshape(tensor=tf.concat(values=[valsn, embed], axis=-1), shape=[batch, dim])

with tf.name_scope(name="deep-part"):
    ydeep =x
    for l in range(len(deep_hidden_sizes)):
        # -----The order for each hidden layer is: matmul => bn => relu => dropout => matmul => ...
        ydeep =tf.layers.dense(inputs=ydeep,
                               units=deep_hidden_sizes[l],
                               activation=None,
                               use_bias=use_deep_hidden_bias,
                               kernel_initializer=xavier_initializer(uniform=True, seed=seed, dtype=dtype),
                               bias_initializer=tf.zeros_initializer(dtype=dtype),
                               kernel_regularizer=l2_regularizer(scale=lamb),
                               bias_regularizer=None,
                               name="deep-dense-hidden-{}".format(l))
        if use_bn ==True:
            ydeep =tf.layers.batch_normalization(inputs=ydeep,
                                                 axis=-1,
                                                 momentum=0.99, # *manual optional*
                                                 epsilon=1e-3, # *manual optional*
                                                 center=True,
                                                 scale=True,
                                                 beta_initializer=tf.zeros_initializer(dtype=dtype),
                                                 gamma_initializer=tf.ones_initializer(dtype=dtype),
                                                 moving_mean_initializer=tf.zeros_initializer(dtype=dtype),
                                                 moving_variance_initializer=tf.ones_initializer(dtype=dtype),
                                                 beta_regularizer=None, # *manual optional*
                                                 gamma_regularizer=None, # *manual optional*
                                                 training=(mode ==tf.estimator.ModeKeys.TRAIN),
                                                 name="dense-bn-hidden-{}".format(l))
        ydeep =tf.nn.relu(features=ydeep)
        if dropouts !=None:
            ydeep =tf.layers.dropout(inputs=ydeep,
                                    rate=dropouts[l],
                                    seed=seed,
                                    training=(mode ==tf.estimator.ModeKeys.TRAIN))

with tf.name_scope(name="cross-part"):
    Wc =tf.get_variable(name="Wc",
                        shape=[num_cross_hidden_layers, dim],
                        dtype=dtype,
                        initializer=xavier_initializer(uniform=False, seed=seed, dtype=dtype),
                        regularizer=l2_regularizer(scale=lamb))
    bc =tf.get_variable(name="bc",
                        shape=[num_cross_hidden_layers, dim],
                        dtype=dtype,

```

```

        initializer=xavier_initializer(uniform=False, seed=seed, dtype=dtype),
        regularizer=l2_regularizer(scale=lamb))

    ycross =x # A tensor in shape of (batch, dim)
    for l in range(num_cross_hidden_layers):
        wl =tf.expand_dims(input=Wc[l], axis=1) # A tensor in shape of (dim, 1)
        bl =bc[l] # A tensor in shape of (dim)
        xwl =tf.matmul(a=ycross, b=wl) # A tensor in shape of (batch, 1)
        ycross =tf.multiply(x=x, y=xwl) +ycross +bl

    with tf.name_scope(name="combine-output"):
        y =tf.concat(values=[ycross, ydeep], axis=-1) # A tensor in shape of (batch, concat_size)
        logits =tf.layers.dense(inputs=y,
                               units=output_size,
                               activation=None,
                               use_bias=use_global_bias,
                               kernel_initializer=xavier_initializer(uniform=True, seed=seed, dtype=dtype),
                               bias_initializer=tf.zeros_initializer(dtype=dtype),
                               kernel_regularizer=l2_regularizer(scale=lamb),
                               bias_regularizer=None,
                               name="output") # A tensor in shape of (batch, output_size)

    if task == "binary":
        logits =tf.squeeze(input=logits, axis=1) # A tensor in shape of (None)
        probs =tf.nn.sigmoid(x=logits)
        classes =tf.cast(x=tf.greater(x=tf.nn.sigmoid(x=logits), y=threshold), dtype=tf.int32)
        predictions ={  

            name_probability_output: probs,  

            name_classification_output: classes  

        }
    elif task == "multi":
        probs_dist =tf.nn.softmax(logits=logits, axis=-1)
        classes =tf.argmax(input=logits, axis=-1, output_type=tf.int32)
        predictions ={  

            name_probability_output: probs_dist,  

            name_classification_output: classes  

        }
    elif task == "regression":
        logits =tf.squeeze(input=logits, axis=1) # A tensor in shape of (None)
        predictions ={  

            name_regression_output: logits  

        }
    else:
        raise ValueError(value_error_warning_task)

# -----Provide an estimator spec for `ModeKeys.PREDICTION` mode-----
if mode ==tf.estimator.ModeKeys.PREDICT:
    export_outputs ={  

        tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:  

            tf.estimator.export.PredictOutput(outputs=predictions)  

    } # For usage of tensorflow serving
    return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions, export_outputs=export_outputs)

```

```

# -----Build loss function-----
if task == "binary":
    loss = tf.reduce_mean(input_tensor=tf.nn.sigmoid_cross_entropy_with_logits(labels=labels,
                                                                           logits=logits),
                           axis=0,
                           keepdims=False) # A scalar, representing the training loss of current batch training
                                                               dataset
elif task == "regression":
    loss = tf.reduce_mean(input_tensor=tf.square(x=tf.subtract(x=labels, y=logits)),
                           axis=0,
                           keepdims=False) # A scalar, representing the training loss of current batch training
                                                               dataset
elif task == "multi":
    labels_one_hot = tf.one_hot(indices=tf.cast(x=labels, dtype=tf.int32),
                                depth=output_size,
                                axis=-1,
                                dtype=dtype) # A tensor in shape of (None, output_size)
    loss = tf.reduce_mean(input_tensor=tf.nn.softmax_cross_entropy_with_logits_v2(labels=labels_one_hot,
                                                                                 logits=logits),
                           axis=0,
                           keepdims=False) # A scalar, representing the training loss of current batch training
                                                               dataset
else:
    raise ValueError(value_error_warning_task)

reg = tf.reduce_sum(input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
                     axis=0,
                     keepdims=False,
                     name="regularization") # A scalar, representing the regularization loss of current batch
                                         training dataset
loss += reg

# -----Provide an estimator spec for `ModeKeys.EVAL` mode-----
if mode == tf.estimator.ModeKeys.EVAL:
    if task == "binary":
        eval_metric_ops = {
            "accuracy": tf.metrics.accuracy(labels=labels,
                                              predictions=predictions[name_classification_output]),
            "precision": tf.metrics.precision(labels=labels,
                                              predictions=predictions[name_classification_output]),
            "recall": tf.metrics.recall(labels=labels, predictions=predictions[name_classification_output]),
            "auc": tf.metrics.auc(labels=labels, predictions=predictions[name_classification_output])
        }
    elif task == "multi":
        eval_metric_ops = {
            "confusion-matrix": tf.confusion_matrix(labels=labels,
                                                      predictions=predictions[name_classification_output],
                                                      num_classes=output_size)
        }
    elif task == "regression":
        eval_metric_ops = {
            "rmse": tf.metrics.root_mean_squared_error(labels=labels,
                                                       predictions=predictions[name_classification_output])
        }

```

```

        predictions=predictions[name_regression_output])

    }

else:
    raise ValueError(value_error_warning_task)

return tf.estimator.EstimatorSpec(mode=mode, loss=loss, predictions=predictions,
                                  eval_metric_ops=eval_metric_ops)

# -----Build optimizer-----
global_step =tf.train.get_or_create_global_step(graph=tf.get_default_graph()) # Define a global step for
                                training step counter

if optimizer == "sgd":
    opt_op =tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
elif optimizer == "sgd-exp-decay":
    decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)
    opt_op =tf.train.GradientDescentOptimizer(learning_rate=decay_learning_rate)
elif optimizer == "momentum":
    opt_op =tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                       momentum=0.9,
                                       use_nesterov=False)
elif optimizer == "momentum-exp-decay":
    decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)
    opt_op =tf.train.MomentumOptimizer(learning_rate=decay_learning_rate,
                                       momentum=0.9,
                                       use_nesterov=False)
elif optimizer == "nesterov":
    opt_op =tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                       momentum=0.9,
                                       use_nesterov=True)
elif optimizer == "nesterov-exp-decay":
    decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)
    opt_op =tf.train.MomentumOptimizer(learning_rate=decay_learning_rate,
                                       momentum=0.9,
                                       use_nesterov=True)
elif optimizer == "adagrad":
    opt_op =tf.train.AdagradOptimizer(learning_rate=learning_rate,
                                      initial_accumulator_value=0.1)
elif optimizer == "adadelta":
    opt_op =tf.train.AdadeltaOptimizer(learning_rate=learning_rate,
                                       rho=0.95)
elif optimizer == "rmsprop":

```

```

    opt_op =tf.train.RMSPropOptimizer(learning_rate=learning_rate,
                                      decay=0.9)
  elif optimizer == "adam":
    opt_op =tf.train.AdamOptimizer(learning_rate=learning_rate,
                                  beta1=0.9,
                                  beta2=0.999)
  else:
    raise NotImplementedError(value_error_warning_optimizer)

train_op =opt_op.minimize(loss=loss, global_step=global_step, name="train_op")

# -----Provide an estimator spec for `ModeKeys.TRAIN` mode-----
if (mode ==tf.estimator.ModeKeys.TRAIN):
  return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

def main(unused_argv):
  # -----Declare all hyperparameters of terminal interface-----
  phase =FLAGS.phase
  model_dir =FLAGS.model_dir
  export_dir =FLAGS.export_dir
  perform_valid_during_train =FLAGS.perform_valid_during_train # (optional) Note: if use validation dataset
                                                               during train phase, the valid set and train set are
                                                               in same data_dir
  log_step_count_steps =FLAGS.log_step_count_steps # (optional)
  save_checkpoints_steps =FLAGS.save_checkpoints_steps # (optional)
  keep_checkpoint_max =FLAGS.keep_checkpoint_max # (optional)
  data_dir =FLAGS.data_dir
  delimiter =FLAGS.delimiter
  field_size_numerical =FLAGS.field_size_numerical
  field_size_categorical =FLAGS.field_size_categorical
  batch_size =FLAGS.batch_size
  epochs =FLAGS.epochs
  shuffle =FLAGS.shuffle # (optional)
  buffer_size =FLAGS.buffer_size # (optional)
  num_parallel_calls =FLAGS.num_parallel_calls # (optional)
  use_dtype_high_precision =FLAGS.use_dtype_high_precision # (optional)
  name_feat_vals_numerical =FLAGS.name_feat_vals_numerical # (optional)
  name_feat_inds_categorical =FLAGS.name_feat_inds_categorical # (optional)
  task =FLAGS.task
  output_size =FLAGS.output_size
  feat_size_categorical =FLAGS.feat_size_categorical
  embed_size =FLAGS.embed_size
  num_cross_hidden_layers =FLAGS.num_cross_hidden_layers
  deep_hidden_sizes =FLAGS.deep_hidden_sizes
  dropouts =FLAGS.dropouts
  use_global_bias =FLAGS.use_global_bias # (optional)
  use_deep_hidden_bias =FLAGS.use_deep_hidden_bias # (optional)
  use_bn =FLAGS.use_bn # (optional)
  lamb =FLAGS.lamb
  optimizer =FLAGS.optimizer
  learning_rate =FLAGS.learning_rate

```

```

PREFIX_TRAIN_FILE ="train"
PREFIX_EVAL_FILE ="eval"
PREFIX_PREDICT_FILE ="predict"
REUSE =False
SEED =None

if use_dtype_high_precision ==False:
    dtype =tf.float32
else:
    dtype =tf.float64

hparams ={
    "task": task,
    "output_size": output_size,
    "field_size_numerical": field_size_numerical,
    "field_size_categorical": field_size_categorical,
    "feat_size_categorical": feat_size_categorical,
    "embed_size": embed_size,
    "num_cross_hidden_layers": num_cross_hidden_layers,
    "deep_hidden_sizes": deep_hidden_sizes,
    "dropouts": dropouts,
    "use_global_bias": use_global_bias,
    "use_deep_hidden_bias": use_deep_hidden_bias,
    "use_bn": use_bn,
    "lamb": lamb,
    "optimizer": optimizer,
    "learning_rate": learning_rate,
    "dtype": dtype,
    "name_feat_vals_numerical": name_feat_vals_numerical,
    "name_feat_inds_categorical": name_feat_inds_categorical,
    "reuse": REUSE,
    "seed": SEED
}
config =tf.estimator.RunConfig(
    log_step_count_steps=log_step_count_steps,
    save_checkpoints_steps=save_checkpoints_steps,
    keep_checkpoint_max=keep_checkpoint_max
)
estimator =tf.estimator.Estimator(model_fn=model_fn, model_dir=model_dir, params=hparams, config=config)
if phase == "train":
    filenames_train =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_TRAIN_FILE +"\*"))
    if perform_valid_during_train ==True:
        filenames_valid =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_EVAL_FILE +"\*"))
        train_spec =tf.estimator.TrainSpec(input_fn=lambda :input_fn(filenames=filenames_train,
                                                                     delimiter=delimiter,
                                                                     field_size_numerical=field_size_numerical,
                                                                     field_size_categorical=field_size_categorical,
                                                                     batch_size=batch_size,
                                                                     epochs=epochs,
                                                                     shuffle=shuffle,
                                                                     buffer_size=buffer_size,
                                                                     num_parallel_calls=num_parallel_calls,

```

```

        dtype=dtype,
        name_feat_vals_numerical=name_feat_vals_numerical,
        name_feat_inds_categorical=name_feat_inds_categorical),
    max_steps=None)
eval_spec = tf.estimator.EvalSpec(input_fn=lambda :input_fn(filenames=filenames_valid,
    delimiter=delimiter,
    field_size_numerical=field_size_numerical,
    field_size_categorical=field_size_categorical,
    batch_size=batch_size,
    epochs=1,
    shuffle=False,
    buffer_size=buffer_size,
    num_parallel_calls=num_parallel_calls,
    dtype=dtype,
    name_feat_vals_numerical=name_feat_vals_numerical,
    name_feat_inds_categorical=name_feat_inds_categorical),
    steps=None,
    start_delay_secs=120,
    throttle_secs=600)
tf.estimator.train_and_evaluate(estimator=estimator, train_spec=train_spec, eval_spec=eval_spec)
else:
    estimator.train(input_fn=lambda :input_fn(filenames=filenames_train,
        delimiter=delimiter,
        field_size_numerical=field_size_numerical,
        field_size_categorical=field_size_categorical,
        batch_size=batch_size,
        epochs=epochs,
        shuffle=shuffle,
        buffer_size=buffer_size,
        num_parallel_calls=num_parallel_calls,
        dtype=dtype,
        name_feat_vals_numerical=name_feat_vals_numerical,
        name_feat_inds_categorical=name_feat_inds_categorical),
        steps=None,
        max_steps=None)
elif phase == "eval":
    filenames_eval = tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_EVAL_FILE + "*"))
    estimator.evaluate(input_fn=lambda :input_fn(filenames=filenames_eval,
        delimiter=delimiter,
        field_size_numerical=field_size_numerical,
        field_size_categorical=field_size_categorical,
        batch_size=batch_size,
        epochs=1,
        shuffle=False,
        buffer_size=buffer_size,
        num_parallel_calls=num_parallel_calls,
        dtype=dtype,
        name_feat_vals_numerical=name_feat_vals_numerical,
        name_feat_inds_categorical=name_feat_inds_categorical))
elif phase == "predict":
    filenames_predict = tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_PREDICT_FILE + "*"))
    p = estimator.predict(input_fn=lambda :input_fn(filenames=filenames_predict,

```

```

        delimiter=delimiter,
        field_size_numerical=field_size_numerical,
        field_size_categorical=field_size_categorical,
        batch_size=batch_size,
        epochs=1,
        shuffle=False,
        buffer_size=buffer_size,
        num_parallel_calls=num_parallel_calls,
        dtype=dtype,
        name_feat_vals_numerical=name_feat_vals_numerical,
        name_feat_inds_categorical=name_feat_inds_categorical))

# -----Usage demo, still need to be accomplished-----
for ele in p:
    print(ele)
elif phase == "export":
    features ={
        name_feat_vals_numerical: tf.placeholder(dtype=dtype, shape=[None, field_size_numerical],
                                                   name=name_feat_vals_numerical),
        name_feat_inds_categorical: tf.placeholder(dtype=tf.int32, shape=[None, field_size_categorical],
                                                   name=name_feat_inds_categorical)
    }
    serving_input_receiver_fn =tf.estimator.export.build_raw_serving_input_receiver_fn(features=features)
    estimator.export_savedmodel(export_dir_base=export_dir,
                               serving_input_receiver_fn=serving_input_receiver_fn)
else:
    raise NotImplementedError("Argument <phase> value: {} is not supported.".format(phase))

if __name__ == '__main__':
    tf.logging.set_verbosity(v=tf.logging.INFO)
    tf.app.run(main=main)

```

54.6 DeepFM

54.6.1 Introduction

本节来自论文：DeepFM: A Factorization-Machine based Neural Network for CTR Prediction[?]

54.6.2 Model formulation

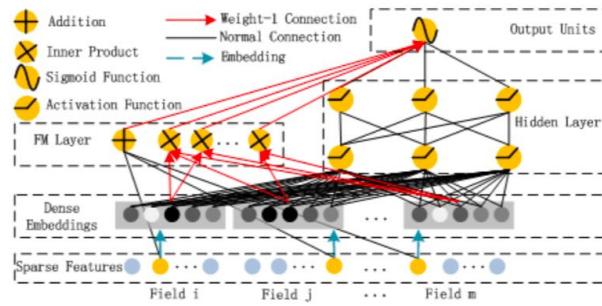


图 178: The structure of DeepFM[?]

Model inference

54.6.3 Model implement

```

tensorflow version
"""
Created on 2019/03/22 16:35
author: Tong Jia
email: cecilio.jia@gmail.com
description:
    An implementation of DeepFM: A Factorization-Machine based Neural Network for CTR Prediction (DeepFM).
    See algorithm and hyperparameter details:
        [Guo et al., 2017] (https://www.ijcai.org/proceedings/2017/0239.pdf)
    The algorithm is developed with TensorFlow Estimator based on TensorFlow 1.12.0 version.
"""

import tensorflow as tf
from tensorflow.contrib.layers import xavier_initializer, l2_regularizer
import os

FLAGS = tf.flags.FLAGS
# -----Hyperparameters of estimator-----
tf.flags.DEFINE_enum(name="phase",
                     default=None,
                     enum_values=["train", "eval", "predict", "export"],
                     help="A string, representing the phase of estimator")
tf.flags.DEFINE_string(name="model_dir",
                      default=None,
                      help="A string, representing the folder path of saved model files")
tf.flags.DEFINE_string(name="export_dir",
                      default=None,
                      help="A string, representing the basic folder path of export .pb files")
# -----Hyperparameters of estimator (optional)-----
tf.flags.DEFINE_boolean(name="perform_valid_during_train",
                       default=True,
                       help="(optional) A boolean, instructing whether to perform validation on valid dataset
                             during train phase")
tf.flags.DEFINE_integer(name="log_step_count_steps",
                       default=500,
                       help="(optional) An integer, representing the frequency, in number of global steps, that
                             the global step/sec will be
                             logged during training",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="save_checkpoints_steps",
                       default=20000,
                       help="(optional) An integer, representing save checkpoints every this many steps",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="keep_checkpoint_max",
                       default=2,
                       help="(optional) An integer, representing max number of saved model checkpoints",
                       lower_bound=1,
                       upper_bound=None)
# -----Hyperparameters of input function-----
tf.flags.DEFINE_string(name="data_dir",

```

```

        default=None,
        help="A string, representing the folder path of dataset")
tf.flags.DEFINE_string(name="delimiter",
                      default=None,
                      help="A string, separating consecutive <index j>:<value j> pairs in a line of data file")
tf.flags.DEFINE_string(name="separator",
                      default=None,
                      help="A string, separating feature index(left part) and feature value(right part) in a
                           specific pair")

tf.flags.DEFINE_integer(name="batch_size",
                       default=None,
                       help="An integer, representing the number of consecutive elements of this dataset to
                            combine in a single batch",
                       lower_bound=1,
                       upper_bound=None)
tf.flags.DEFINE_integer(name="epochs",
                       default=None,
                       help="An integer, representing the number of times the dataset should be repeated",
                       lower_bound=1,
                       upper_bound=None)

# -----Hyperparameters of input function (optional)-----
tf.flags.DEFINE_boolean(name="shuffle",
                        default=True,
                        help="(optional) A boolean, instructing whether to randomly shuffle the samples of
                             training dataset")

tf.flags.DEFINE_integer(name="buffer_size",
                       default=100000,
                       help="(optional) An integer scalar, denoting the number of bytes to buffer",
                       lower_bound=1,
                       upper_bound=None)

tf.flags.DEFINE_integer(name="num_parallel_calls",
                       default=4,
                       help="(optional) An integer scalar, representing the number elements to process in
                            parallel",
                       lower_bound=1,
                       upper_bound=None)

tf.flags.DEFINE_boolean(name="use_dtype_high_precision",
                       default=False,
                       help="(optional) A boolean, instructing the dtype of both input function and model
                            function. If False, use
                            tf.float32; if True, use
                            tf.float64")

tf.flags.DEFINE_string(name="name_feat_inds",
                      default="inds",
                      help="(optional) A string, representing the name of feature indices in return dict")
tf.flags.DEFINE_string(name="name_feat_vals",
                      default="vals",
                      help="(optional) A string, representing the name of feature values in return dict")

# -----Hyperparameters of model function-----
tf.flags.DEFINE_string(name="task",
                      default=None,
                      help="A string, representing the type of task (binary or regression)")

```

```

tf.flags.DEFINE_integer(name="field_size",
                       default=None,
                       help="An integer scalar, representing the number of fields(number of columns before
                             one-hot encoding) of dataset")

tf.flags.DEFINE_integer(name="feat_size",
                       default=None,
                       help="An integer scalar, representing the number of features(number of columns after
                             one-hot encoding) of dataset")

tf.flags.DEFINE_integer(name="embed_size",
                       default=None,
                       help="An integer scalar, representing the dimension of embedding vectors for all
                             features")

tf.flags.DEFINE_list(name="hidden_sizes",
                     default=None,
                     help="A list, containing the number of hidden units of each hidden layer in dnn part")

tf.flags.DEFINE_list(name="dropouts",
                     default=None,
                     help="A list or None, containing the dropout rate of each hidden layer in dnn part "
                          "(e.g. 0.4,0.3,0.2,0.1); if None, don't use dropout operation for any hidden layer")

# -----Hyperparameters of model function (optional)-----
tf.flags.DEFINE_boolean(name="use_global_bias",
                        default=True,
                        help="(optional) A boolean, instructing whether to use global bias in model inference")

tf.flags.DEFINE_boolean(name="use_linear",
                        default=True,
                        help="(optional) A boolean, instructing whether to use linear part in model inference")

tf.flags.DEFINE_boolean(name="use_hidden_bias",
                        default=True,
                        help="(optional) A boolean, instructing whether to use bias of hidden layer units in
                              model inference")

tf.flags.DEFINE_boolean(name="use_bn",
                        default=True,
                        help="(optional) A boolean, instructing whether to use batch normalization for each
                              hidden layer in dnn part")

tf.flags.DEFINE_float(name="lamb",
                      default=0.001,
                      help="(optional) A float scalar, representing the coefficient of regularization term",
                      lower_bound=0.0,
                      upper_bound=None)

tf.flags.DEFINE_string(name="optimizer",
                      default="adam",
                      help="(optional) A string, representing the type of optimizer")

tf.flags.DEFINE_float(name="learning_rate",
                      default=0.003,
                      help="(optional) A float scalar, representing the learning rate of optimizer",
                      lower_bound=0.0,
                      upper_bound=None)

def input_fn(filenames,
             delimiter,
             separator,
             batch_size,

```

```

    epochs,
    shuffle=True,
    buffer_size=100000,
    num_parallel_calls=4,
    dtype=tf.float32,
    name_feat_inds="inds",
    name_feat_vals="vals"):

"""
The input function for loading multi-field sparse dataset. The columns are
separated by argument delimiter(e.g. " ") with the following schema (libsvm format):
<label> <index 1>:<value 1> ... <index j>:<value j> ... <index d>:<value d> (d is the field size of dataset)
e.g.
0 0:0.732349 2:1 41:1 59:1 66:1 77:1 85:1 88:1 89:1 92:1 93:1 98:1 106:1
where:
delimiter is always " " in txt file format, "," in csv file format;
separator is always equal to ":".

Note:
1. The input function is only used for dataset with one-hot active value in each field.
2. In each line, the order of all fields must be fixed.
3. The input function can be used for binary classification and regression task:
binary classification: <label> in {0, 1};
regression: <label> in (-inf, inf).

Parameters
-----
:param filenames: list
    A list of string, containing one or more paths of filenames.
:param delimiter: str
    A str, separating consecutive <index j>:<value j> pairs in data files.
:param separator: str
    A str, separating feature index(left part of key-value pair) and feature value(right part of key-value
    pair) in one specific pair.
:param batch_size: int
    An integer scalar, representing the number of consecutive elements of this dataset to combine in a
    single batch.
:param epochs: int
    An integer scalar, representing the number of times the dataset should be repeated.
:param shuffle: bool, optional
    A boolean(defaults to True), instructing whether to randomly shuffle the elements of this dataset.
:param buffer_size: int, optional
    An integer scalar(defaults to 100000), denoting the number of bytes to buffer.
:param num_parallel_calls: int, optional
    An integer scalar(defaults to 4), representing the number elements to process in parallel.
:param dtype: tf.Dtype, optional
    A tf.DType(defaults to tf.float32), representing the numeric type of values. it always takes value from
    [tf.float32, tf.float64].
:param name_feat_inds: str, optional
    A string, representing the name of feature indices in return dict.
:param name_feat_vals: str, optional
    A string, representing the name of feature values in return dict.

Returns

```

```

-----
:return: dict
    A dict of two Tensors, representing features(including feature indices and feature values) in a single
    batch.
{
    <name_feat_inds>: tf.Tensor of feature indices in shape of (None, field_size),
    <name_feat_vals>: tf.Tensor of feature values in shape of (None, field_size)
}
:return: Tensor
    A Tensor in shape of (None), representing labels in a single batch.
"""

def map_func(line):
    columns =tf.string_split(source=[line], delimiter=delimiter, skip_empty=False).values
    label =tf.string_to_number(string_tensor=columns[0], out_type=dtype)
    splits =tf.string_split(source=columns[1: ], delimiter=separator, skip_empty=False)
    feats =tf.reshape(
        tensor=splits.values, shape=splits.dense_shape
    ) # A tensor in shape of (field_size, 2), the first column contains feature indices, the second column
        contains feature values
    inds, vals =tf.split(value=feats, num_or_size_splits=2, axis=1) # Two tensors in shape of (field_size,
        1)
    inds =tf.reshape(tensor=tf.string_to_number(string_tensor=inds, out_type=tf.int32), shape=[-1]) # A
        tensor in shape of (field_size)
    vals =tf.reshape(tensor=tf.string_to_number(string_tensor=vals, out_type=dtype), shape=[-1]) # A
        tensor in shape of (field_size)
    return {name_feat_inds: inds, name_feat_vals: vals}, label

dataset =tf.data.TextLineDataset(filenames=filenames, buffer_size=buffer_size). \
    map(map_func=map_func, num_parallel_calls=num_parallel_calls)
if (shuffle ==True):
    dataset =dataset.shuffle(buffer_size=buffer_size)
dataset =dataset. \
    repeat(count=epochs). \
    batch(batch_size=batch_size, drop_remainder=False)
dataset =dataset.prefetch(buffer_size=1)
iterator =dataset.make_one_shot_iterator()
batch_feats, batch_labels =iterator.get_next()
return batch_feats, batch_labels

def model_fn(features, labels, mode, params):
    """Model function of DeepFM for predictive analytics of high dimensional sparse data.

    Args of dict params:
        task: str
            A string, representing the type of task.
        Note:
            it must take value from ["binary", "regression"];
            it instruct the type of loss function:
            "binary": sigmoid cross-entropy;
            "regression": mean squared error.
        field_size: int
    """

```

```

    An integer scalar, representing the number of fields(number of columns before one-hot encoding) of
    dataset.

feat_size: int
    An integer scalar, representing the number of features(number of columns after one-hot encoding) of
    dataset.

embed_size: int
    An integer scalar, representing the dimension of embedding vectors for all features.

hidden_sizes: list
    A list, containing the number of hidden units of each hidden layer in dnn part.

    Note:
        it doesn't contain output layer of dnn part.

dropouts: list or None
    If list, containing the dropout rate of each hidden layer in dnn part;
    If None, don't use dropout operation for any hidden layer.

    Note:
        if list, the length of <dropouts> must be equal to <hidden_sizes>.

use_global_bias: bool
    A boolean, instructing whether to use global bias in model inference.

use_linear: bool
    A boolean, instructing whether to use linear part in model inference.

use_hidden_bias: bool
    A boolean, instructing whether to use bias of hidden layer units in model inference.

use_bn: bool
    A boolean, instructing whether to use batch normalization for each hidden layer in dnn part.

lamb: float
    A float scalar, representing the coefficient of regularization term (the larger the value of lamb,
    the stronger the penalty is).

optimizer: str
    A string, representing the type of optimizer.

learning_rate: float
    A float scalar, representing the learning rate of optimizer.

dtype: tf.Dtype
    A tf.DType, representing the numeric type of values.

    Note:
        it must take value from [tf.float32, tf.float64];
        it must be consistent with <dtype> of input function.

name_feat_inds: str
    A string, representing the name of feature indices in return dict.

    Note:
        it must be consistent with <name_feat_inds> of input function.

name_feat_vals: str
    A string, representing the name of feature values in return dict.

    Note:
        it must be consistent with <name_feat_vals> of input function.

reuse: bool
    A boolean, which takes value from [False, True, tf.AUTO_REUSE].

seed: int or None
    If integer scalar, representing the random seed of tensorflow;
    If None, random choice.

"""
# -----Declare all hyperparameters from params-----
task = params["task"]

```

```

field_size =params["field_size"]
feat_size =params["feat_size"]
embed_size =params["embed_size"]
hidden_sizes =params["hidden_sizes"]
dropouts =params["dropouts"]
use_global_bias =params["use_global_bias"]
use_linear =params["use_linear"]
use_hidden_bias =params["use_hidden_bias"]
use_bn =params["use_bn"]
lamb =params["lamb"]
optimizer =params["optimizer"]
learning_rate =params["learning_rate"]
dtype =params["dtype"]
name_feat_inds =params["name_feat_inds"]
name_feat_vals =params["name_feat_vals"]
reuse =params["reuse"]
seed =params["seed"]
# -----Hyperparameters for threshold for binary classification task
threshold =0.5
# -----Hyperparameters for exponential decay(*manual optional*)-----
decay_steps =5000
decay_rate =0.998
staircase =True
# -----Hyperparameters for information showing-----
name_probability_output ="prob"
name_classification_output ="class"
name_regression_output ="pred"
value_error_warning_task ="Argument of model function <task>: \"{}\" is invalid. It must be in
                                         [\"binary\", \"regression\"]".format(task)
value_error_warning_optimizer ="Argument value of <optimizer>: {} is not supported.".format(optimizer)

if seed !=None:
    tf.set_random_seed(seed)

# -----Assert for hyperparameters-----
assert (task in ["binary", "regression"])
if dropouts !=None:
    assert (len(dropouts) ==len(hidden_sizes))
    assert (dtype in [tf.float32, tf.float64])

# -----Build model inference-----
with tf.variable_scope(name_or_scope="inference", reuse=reuse):
    with tf.name_scope(name="inputs"):
        ids =features[name_feat_inds]
        x =features[name_feat_vals]

    with tf.name_scope(name="global-bias"):
        b =tf.get_variable(name="b",
                           shape=[1],
                           dtype=dtype,
                           initializer=tf.zeros_initializer(dtype=dtype),
                           regularizer=l2_regularizer(scale=lamb),

```

```

        trainable=use_global_bias)

with tf.name_scope(name="lr-part"):
    W = tf.get_variable(name="W",
                        shape=[feat_size],
                        dtype=dtype,
                        initializer=xavier_initializer(uniform=True, dtype=dtype), # *manual optional*
                        regularizer=l2_regularizer(scale=lamb),
                        trainable=use_linear)
    # -----embedding lookup op for first order weights-----
    w = tf.nn.embedding_lookup(params=W, ids=ids) # A tensor in shape of (None, field_size)
    ylr = tf.reduce_sum(input_tensor=tf.multiply(x=x, y=w), # A tensor in shape of (None, field_size)
                        axis=1,
                        keepdims=False) # A tensor in shape of (None)

with tf.name_scope(name="fm-part"):
    V = tf.get_variable(name="V",
                        shape=[feat_size, embed_size],
                        dtype=dtype,
                        initializer=xavier_initializer(uniform=False, dtype=dtype), # manual optional
                        regularizer=None, # *manual optional*
                        trainable=True)
    # -----embedding lookup op for second order weights-----
    v = tf.nn.embedding_lookup(params=V, ids=ids) # A tensor in shape of (None, field_size, embed_size)
    xreshape = tf.expand_dims(input=x, axis=-1) # A tensor in shape of (None, field_size, 1)
    vx = tf.multiply(x=xreshape, y=v) # A tensor in shape of (None, field_size, embed_size)
    p = tf.square(x=tf.reduce_sum(input_tensor=vx, axis=1, keepdims=False)) # a tensor in shape of
                                # (None, embedding_size)
    q = tf.reduce_sum(input_tensor=tf.square(x=vx), # A tensor in shape of (None, field_size,
                      embedding_size)
                      axis=1,
                      keepdims=False) # A tensor in shape of (None, embedding_size)
    yfm = 0.5 *tf.reduce_sum(input_tensor=tf.subtract(x=p, y=q), # A tensor in shape of (None,
                                embedding_size)
                                axis=1,
                                keepdims=False) # A tensor in shape of (None)

with tf.name_scope(name="mlp-part"):
    ymlp =tf.reshape(tensor=vx, shape=[-1, field_size *embed_size]) # a tensor in shape of (None,
                                                                    field_size * embed_size)
    for l in range(len(hidden_sizes)):
        # -----The order for each hidden layer is: matmul => bn => relu => dropout => matmul => ...
        ymlp =tf.layers.dense(inputs=ymlp,
                              units=hidden_sizes[l],
                              activation=None,
                              use_bias=use_hidden_bias,
                              kernel_initializer=xavier_initializer(uniform=True, dtype=dtype),
                              bias_initializer=tf.zeros_initializer(dtype=dtype),
                              kernel_regularizer=None, # *manual optional*
                              bias_regularizer=None, # *manual optional*
                              trainable=True,
                              name="mlp-dense-hidden-{}".format(l))

```

```

if use_bn ==True:
    ymlp =tf.layers.batch_normalization(inputs=ymlp,
                                         axis=-1,
                                         momentum=0.99, # *manual optional*
                                         epsilon=1e-3, # *manual optional*
                                         center=True,
                                         scale=True,
                                         beta_initializer=tf.zeros_initializer(dtype=dtype),
                                         gamma_initializer=tf.ones_initializer(dtype=dtype),
                                         moving_mean_initializer=tf.zeros_initializer(dtype=dtype),
                                         moving_variance_initializer=tf.ones_initializer(dtype=dtype),
                                         beta_regularizer=None, # *manual optional*
                                         gamma_regularizer=None, # *manual optional*
                                         training=(mode ==tf.estimator.ModeKeys.TRAIN),
                                         trainable=True,
                                         name="mlp-bn-hidden-{}".format(l))

ymlp =tf.nn.relu(features=ymlp)
if dropouts !=None:
    ymlp =tf.layers.dropout(inputs=ymlp,
                           rate=dropouts[1],
                           seed=seed,
                           training=(mode ==tf.estimator.ModeKeys.TRAIN))
# -----output layer of mlp part-----
ymlp =tf.layers.dense(inputs=ymlp,
                      units=1,
                      activation=None,
                      use_bias=False,
                      kernel_initializer=xavier_initializer(uniform=True, dtype=dtype),
                      kernel_regularizer=None, # *manual optional*
                      trainable=True,
                      name="mlp-dense-output") # A tensor in shape of (None, 1)
ymlp =tf.squeeze(input=ymlp, axis=1) # A tensor in shape of (None)

with tf.name_scope(name="output"):
    if use_global_bias ==False and use_linear ==False:
        logits =yfm +ymlp
    elif use_global_bias ==False and use_linear ==True:
        logits =ylr +yfm +ymlp
    elif use_global_bias ==True and use_linear ==False:
        logits =b +yfm +ymlp
    else:
        logits =b +ylr +yfm +ymlp

    if task == "binary":
        logits =tf.squeeze(input=logits, axis=1) # A tensor in shape of (None)
        probs =tf.nn.sigmoid(x=logits)
        classes =tf.cast(x=tf.greater(x=tf.nn.sigmoid(x=logits), y=threshold), dtype=tf.int32)
        predictions ={  
            name_probability_output: probs,  
            name_classification_output: classes  
        }

```

```

        elif task == "regression":
            logits = tf.squeeze(input=logits, axis=1) # A tensor in shape of (None)
            predictions = {
                name_regression_output: logits
            }
        else:
            raise ValueError(value_error_warning_task)

# -----Provide an estimator spec for `ModeKeys.PREDICTION` mode-----
if mode == tf.estimator.ModeKeys.PREDICT:
    export_outputs = {
        tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
            tf.estimator.export.PredictOutput(outputs=predictions)
    } # For usage of tensorflow serving
    return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions, export_outputs=export_outputs)

# -----Build loss function-----
if task == "binary":
    loss = tf.reduce_mean(input_tensor=tf.nn.sigmoid_cross_entropy_with_logits(labels=labels,
                                                                           logits=logits),
                           axis=0,
                           keepdims=False) # A scalar, representing the training loss of current batch training
                           dataset
elif task == "regression":
    loss = tf.reduce_mean(input_tensor=tf.square(x=tf.subtract(x=labels, y=logits)),
                           axis=0,
                           keepdims=False) # A scalar, representing the training loss of current batch training
                           dataset
else:
    raise ValueError(value_error_warning_task)

reg = tf.reduce_sum(input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
                     axis=0,
                     keepdims=False,
                     name="regularization") # A scalar, representing the regularization loss of current batch
                     training dataset
loss += reg

# -----Provide an estimator spec for `ModeKeys.EVAL` mode-----
if mode == tf.estimator.ModeKeys.EVAL:
    if task == "binary":
        eval_metric_ops = {
            "accuracy": tf.metrics.accuracy(labels=labels,
                                             predictions=predictions[name_classification_output]),
            "precision": tf.metrics.precision(labels=labels,
                                              predictions=predictions[name_classification_output]),
            "recall": tf.metrics.recall(labels=labels, predictions=predictions[name_classification_output]),
            "auc": tf.metrics.auc(labels=labels, predictions=predictions[name_classification_output])
        }
    elif task == "regression":
        eval_metric_ops = {
            "rmse": tf.metrics.root_mean_squared_error(labels=labels,
}

```

```

                predictions=predictions[name_regression_output])
            }
        else:
            raise ValueError(value_error_warning_task)
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, predictions=predictions,
                                      eval_metric_ops=eval_metric_ops)

# -----Build optimizer-----
global_step =tf.train.get_or_create_global_step(graph=tf.get_default_graph()) # Define a global step for
                                training step counter
if optimizer == "sgd":
    opt_op =tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
elif optimizer == "sgd-exp-decay":
    decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)
    opt_op =tf.train.GradientDescentOptimizer(learning_rate=decay_learning_rate)
elif optimizer == "momentum":
    opt_op =tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                       momentum=0.9,
                                       use_nesterov=False)
elif optimizer == "momentum-exp-decay":
    decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)
    opt_op =tf.train.MomentumOptimizer(learning_rate=decay_learning_rate,
                                       momentum=0.9,
                                       use_nesterov=False)
elif optimizer == "nesterov":
    opt_op =tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                       momentum=0.9,
                                       use_nesterov=True)
elif optimizer == "nesterov-exp-decay":
    decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
                                                    global_step=global_step,
                                                    decay_steps=decay_steps,
                                                    decay_rate=decay_rate,
                                                    staircase=staircase)
    opt_op =tf.train.MomentumOptimizer(learning_rate=decay_learning_rate,
                                       momentum=0.9,
                                       use_nesterov=True)
elif optimizer == "adagrad":
    opt_op =tf.train.AdagradOptimizer(learning_rate=learning_rate,
                                      initial_accumulator_value=0.1)
elif optimizer == "adadelta":
    opt_op =tf.train.AdadeltaOptimizer(learning_rate=learning_rate,
                                       rho=0.95)
elif optimizer == "rmsprop":

```

```

    opt_op =tf.train.RMSPropOptimizer(learning_rate=learning_rate,
                                      decay=0.9)

    elif optimizer == "adam":
        opt_op =tf.train.AdamOptimizer(learning_rate=learning_rate,
                                       beta1=0.9,
                                       beta2=0.999)

    else:
        raise NotImplementedError(value_error_warning_optimizer)

    train_op =opt_op.minimize(loss=loss, global_step=global_step, name="train_op")

    # -----Provide an estimator spec for `ModeKeys.TRAIN` mode-----
    if (mode ==tf.estimator.ModeKeys.TRAIN):
        return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

def main(unused_argv):
    # -----Declare all hyperparameters of terminal interface-----
    phase =FLAGS.phase
    perform_valid_during_train =FLAGS.perform_valid_during_train # (optional) Note: if use validation dataset
                                                               during train phase, the valid set and train set are
                                                               in same data_dir
    log_step_count_steps =FLAGS.log_step_count_steps # (optional)
    save_checkpoints_steps =FLAGS.save_checkpoints_steps # (optional)
    keep_checkpoint_max =FLAGS.keep_checkpoint_max # (optional)
    model_dir =FLAGS.model_dir
    export_dir =FLAGS.export_dir
    data_dir =FLAGS.data_dir
    delimiter =FLAGS.delimiter
    separator =FLAGS.separator
    batch_size =FLAGS.batch_size
    epochs =FLAGS.epochs
    shuffle =FLAGS.shuffle
    buffer_size =FLAGS.buffer_size # (optional)
    num_parallel_calls =FLAGS.num_parallel_calls # (optional)
    use_dtype_high_precision =FLAGS.use_dtype_high_precision # (optional)
    name_feat_inds =FLAGS.name_feat_inds # (optional)
    name_feat_vals =FLAGS.name_feat_vals # (optional)
    task =FLAGS.task
    field_size =FLAGS.field_size
    feat_size =FLAGS.feat_size
    embed_size =FLAGS.embed_size
    hidden_sizes =FLAGS.hidden_sizes
    dropouts =FLAGS.dropouts
    use_global_bias =FLAGS.use_global_bias # (optional)
    use_linear =FLAGS.use_linear # (optional)
    use_hidden_bias =FLAGS.use_hidden_bias # (optional)
    use_bn =FLAGS.use_bn # (optional)
    lamb =FLAGS.lamb
    optimizer =FLAGS.optimizer
    learning_rate =FLAGS.learning_rate

```

```

PREFIX_TRAIN_FILE ="train"
PREFIX_EVAL_FILE ="eval"
PREFIX_PREDICT_FILE ="predict"
REUSE =False
SEED =None

if use_dtype_high_precision ==False:
    dtype =tf.float32
else:
    dtype =tf.float64
hidden_sizes =[int(float(ele)) for ele in hidden_sizes]
if dropouts !=None:
    dropouts =[float(ele) for ele in dropouts]

hparams ={
    "task": task,
    "field_size": field_size,
    "feat_size": feat_size,
    "embed_size": embed_size,
    "hidden_sizes": hidden_sizes,
    "dropouts": dropouts,
    "use_global_bias": use_global_bias,
    "use_linear": use_linear,
    "use_hidden_bias": use_hidden_bias,
    "use_bn": use_bn,
    "lamb": lamb,
    "optimizer": optimizer,
    "learning_rate": learning_rate,
    "dtype": dtype,
    "name_feat_inds": name_feat_inds,
    "name_feat_vals": name_feat_vals,
    "reuse": REUSE,
    "seed": SEED
}
config =tf.estimator.RunConfig(
    log_step_count_steps=log_step_count_steps,
    save_checkpoints_steps=save_checkpoints_steps,
    keep_checkpoint_max=keep_checkpoint_max
)
estimator =tf.estimator.Estimator(model_fn=model_fn, model_dir=model_dir, params=hparams, config=config)
if phase == "train":
    filenames_train =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_TRAIN_FILE +"*"))
    if perform_valid_during_train ==True:
        filenames_valid =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_EVAL_FILE +"*"))
        train_spec =tf.estimator.TrainSpec(input_fn=lambda :input_fn(filenames=filenames_train,
                                                                     delimiter=delimiter,
                                                                     separator=separator,
                                                                     batch_size=batch_size,
                                                                     epochs=epochs,
                                                                     shuffle=shuffle,
                                                                     buffer_size=buffer_size,
                                                                     num_parallel_calls=num_parallel_calls,

```

```

        dtype=dtype,
        name_feat_inds=name_feat_inds,
        name_feat_vals=name_feat_vals),
    max_steps=None)

eval_spec = tf.estimator.EvalSpec(input_fn=lambda :input_fn(filenames=filenames_valid,
    delimiter=delimiter,
    separator=separator,
    batch_size=batch_size,
    epochs=1,
    shuffle=False,
    buffer_size=buffer_size,
    num_parallel_calls=num_parallel_calls,
    dtype=dtype,
    name_feat_inds=name_feat_inds,
    name_feat_vals=name_feat_vals),
    steps=None,
    start_delay_secs=120,
    throttle_secs=600)

tf.estimator.train_and_evaluate(estimator=estimator, train_spec=train_spec, eval_spec=eval_spec)
else:
    estimator.train(input_fn=lambda :input_fn(filenames=filenames_train,
        delimiter=delimiter,
        separator=separator,
        batch_size=batch_size,
        epochs=epochs,
        shuffle=shuffle,
        buffer_size=buffer_size,
        num_parallel_calls=num_parallel_calls,
        dtype=dtype,
        name_feat_inds=name_feat_inds,
        name_feat_vals=name_feat_vals),
        steps=None,
        max_steps=None)

elif phase == "eval":
    filenames_eval = tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_EVAL_FILE + "*"))
    estimator.evaluate(input_fn=lambda :input_fn(filenames=filenames_eval,
        delimiter=delimiter,
        separator=separator,
        batch_size=batch_size,
        epochs=1,
        shuffle=False,
        buffer_size=buffer_size,
        num_parallel_calls=num_parallel_calls,
        dtype=dtype,
        name_feat_inds=name_feat_inds,
        name_feat_vals=name_feat_vals))

elif phase == "predict":
    filenames_predict = tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_PREDICT_FILE + "*"))
    p = estimator.predict(input_fn=lambda :input_fn(filenames=filenames_predict,
        delimiter=delimiter,
        separator=separator,
        batch_size=batch_size,

```

```

        epochs=1,
        shuffle=False,
        buffer_size=buffer_size,
        num_parallel_calls=num_parallel_calls,
        dtype=dtype,
        name_feat_inds=name_feat_inds,
        name_feat_vals=name_feat_vals))

# -----Usage demo, still need to be accomplished-----
for ele in p:
    print(ele)
elif phase == "export":
    features ={
        name_feat_inds: tf.placeholder(dtype=tf.int32, shape=[None, field_size], name=name_feat_inds),
        name_feat_vals: tf.placeholder(dtype=dtype, shape=[None, field_size], name=name_feat_vals)
    }
    serving_input_receiver_fn =tf.estimator.export.build_raw_serving_input_receiver_fn(features=features)
    estimator.export_savedmodel(export_dir_base=export_dir,
                                serving_input_receiver_fn=serving_input_receiver_fn)
else:
    raise NotImplementedError("Argument <phase> value: {} is not supported.".format(phase))

if __name__ == '__main__':
    tf.logging.set_verbosity(v=tf.logging.INFO)
    tf.app.run(main=main)

```

torch version 文件结构如下：

- models
 - data_loader.py
 - net.py
 - objective.py
 - optimizer.py
 - metrics.py
 - utils.py
- options
 - options_base.py
 - options_train.py
 - options_eval.py
 - options_infer.py
- checkpoints
- train.py
- evaluate.py

- inference.py

这里着重介绍核心的 models 文件夹中的代码，其余的参考[Code examples in pyTorch and Tensorflow for CS230](#)

```
data_loader
# Copyright (C) 2019 Tong Jia. All rights reserved.
import codecs

import torch
from torch.utils.data import Dataset, DataLoader
from six.moves import xrange


class FeatKVDatasetTrain(Dataset):
    """Define the dataset with key-value(index-value) feature format for train phase. Yield both batch features
       and labels."""
    def __init__(self, filename, splitter_outside, splitter_inside):
        self.file =codecs.open(filename=filename, mode="r").readlines()
        self.splitter_outside =splitter_outside
        self.splitter_inside =splitter_inside

    def __len__(self):
        return len(self.file)

    def __getitem__(self, idx):
        return self.__parser__(line=self.file[idx])

    def __parser__(self, line):
        """Parser each line in raw data file.

        :param line: (str) -- containing data information of single sample.
                      e.g. "1-0:1-3:1-7:1-14:1-18:0.1210-19:0.2413"
        """
        columns =line.strip().split(self.splitter_outside)
        y =float(columns[0])
        x_idx =[]
        x_val =[]
        for i in xrange(1, len(columns)):
            feat_kv =columns[i].split(self.splitter_inside)
            x_idx.append(float(feat_kv[0]))
            x_val.append(float(feat_kv[1]))
        x_idx =torch.tensor(data=x_idx, dtype=torch.long, requires_grad=False)
        x_val =torch.tensor(data=x_val, dtype=torch.float32, requires_grad=False)
        y =torch.tensor(data=y, dtype=torch.float32, requires_grad=False)
        return {"idx": x_idx, "val": x_val}, y

class FeatKVDatasetEval(Dataset):
    """Define the dataset with key-value(index-value) feature format for evaluate phase. Yield both batch
       features and labels."""
    def __init__(self, filename, splitter_outside, splitter_inside):
        self.file =codecs.open(filename=filename, mode="r").readlines()
```

```

        self.splitter_outside = splitter_outside
        self.splitter_inside = splitter_inside

    def __len__(self):
        return len(self.file)

    def __getitem__(self, idx):
        return self.__parser__(line=self.file[idx])

    def __parser__(self, line):
        """Parser each line in raw data file.

        :param line: (str) -- containing data information of single sample.
                           e.g. "1-0:1-3:1-7:1-14:1-18:0.1210-19:0.2413"
        """
        columns = line.strip().split(self.splitter_outside)
        y = float(columns[0])
        x_idx = []
        x_val = []
        for i in xrange(1, len(columns)):
            feat_kv = columns[i].split(self.splitter_inside)
            x_idx.append(float(feat_kv[0]))
            x_val.append(float(feat_kv[1]))
        x_idx = torch.tensor(data=x_idx, dtype=torch.long, requires_grad=False)
        x_val = torch.tensor(data=x_val, dtype=torch.float32, requires_grad=False)
        y = torch.tensor(data=y, dtype=torch.float32, requires_grad=False)
        return {"idx": x_idx, "val": x_val}, y

class FeatKVDatasetInfer(Dataset):
    """Define the dataset with key-value(index-value) feature format for inference phase. Yield only batch
       features"""

    def __init__(self, filename, splitter_outside, splitter_inside):
        self.file = codecs.open(filename=filename, mode="r").readlines()
        self.splitter_outside = splitter_outside
        self.splitter_inside = splitter_inside

    def __len__(self):
        return len(self.file)

    def __getitem__(self, idx):
        """Parser each line in raw data file.

        :param line: (str) -- containing data information of single sample. (don't contain label in first
                           column)
                           e.g. "0:1-3:1-7:1-14:1-18:0.1210-19:0.2413"
        """
        return self.__parser__(line=self.file[idx])

    def __parser__(self, line):
        columns = line.strip().split(self.splitter_outside)
        x_idx = []

```

```

x_val = []
for i in xrange(1, len(columns)):
    feat_kv = columns[i].split(self.splitter_inside)
    x_idx.append(float(feat_kv[0]))
    x_val.append(float(feat_kv[1]))
x_idx = torch.tensor(data=x_idx, dtype=torch.long, requires_grad=False)
x_val = torch.tensor(data=x_val, dtype=torch.float32, requires_grad=False)
return {"idx": x_idx, "val": x_val}

def fetch_deepfm_dataloader(filename,
                            phase,
                            splitter_outside,
                            splitter_inside,
                            batch_size,
                            shuffle_train_phase,
                            num_workers=3):
    """Fetch the data loader from single data file.

    :param filename: (str) -- representing the path of data file.
    :param phase: (str) -- indicating the phase of model. Valid value in ["train", "eval", "infer"].
    :param splitter_outside: (str) -- representing the splitter between columns in each sample of data file.
    :param splitter_inside: (str) -- representing the splitter between key and value in each column of data
        sample line.
    :param batch_size: (int) -- representing the maximum batch size of data.
    :param shuffle_train_phase: (bool) -- indicating whether to shuffle dataset during train phase.
    :param num_workers: (int) -- representing the number of workers for loading dataset.
    :return: (torch.utils.data.DataLoader) -- representing the dataloader object of corresponding data file.
    """

    if phase == "train":
        dataset = FeatKVDatasetTrain(filename,
                                      splitter_inside=splitter_inside,
                                      splitter_outside=splitter_outside)
        dataloader = DataLoader(dataset=dataset, batch_size=batch_size, shuffle=shuffle_train_phase,
                               num_workers=num_workers)

    elif phase == "eval":
        dataset = FeatKVDatasetEval(filename,
                                    splitter_inside=splitter_inside,
                                    splitter_outside=splitter_outside)
        dataloader = DataLoader(dataset=dataset, batch_size=batch_size, shuffle=False, num_workers=num_workers)

    elif phase == "infer":
        dataset = FeatKVDatasetInfer(filename,
                                     splitter_inside=splitter_inside,
                                     splitter_outside=splitter_outside)
        dataloader = DataLoader(dataset=dataset, batch_size=batch_size, shuffle=False, num_workers=num_workers)

    else:
        raise NotImplementedError("phase [%s] is not supported. Valid value must be in [\\"train\\", \\"eval\\",
                                 \\"infer\\\"]")

    return dataloader

```

```

net
# Copyright (C) 2019 Tong Jia. All rights reserved.
import sys
root_path ="../../../../"
if root_path not in sys.path:
    sys.path.append(root_path)

import torch
import torch.nn as nn

class EmbeddingModule(nn.Module):
    """Define the embedding module of DeepFM model."""
    def __init__(self, feature_size, embedding_size):
        """Initialize the class of embedding module.

        :param feature_size: (int) -- representing the number of features that considered in embedding
                                     operation.
        :param embedding_size: (int) -- representing the dimension of embedding vector.
        """
        super(EmbeddingModule, self).__init__()
        self.embeddings =nn.Embedding(num_embeddings=feature_size, embedding_dim=embedding_size)

    def forward(self, idx, val):
        """Standard forward of the module.

        :param idx: (tensor) -- representing the indices of batch data features with shape of (None,
                               field_size).
        :param val: (tensor) -- representing the values of batch data features with shape of (None, field_size).
        :return: (tensor) -- representing the embedding vectors(multiplied with feature values) of correspond
                           batch features with shape of
                           (None, field_size, embedding_size).
        """
        embeds =self.embeddings.forward(input=idx) # a tensor with shape of (None, field_size, embedding_size)
        vals =val.unsqueeze(dim=-1) # a tensor with shape of (None, field_size, 1)
        return torch.mul(input=embeds, other=vals)

class LinearModule(nn.Module):
    """Define the pure linear(excluding bias) module of DeepFM model."""
    def __init__(self, feature_size):
        """Initialize the class of linear module.

        :param feature_size: (int) -- representing the number of features that considered in linear layer.
        """
        super(LinearModule, self).__init__()
        self.weights =nn.Embedding(num_embeddings=feature_size, embedding_dim=1)

    def forward(self, idx, val):
        """Standard forward of the module.

        :param idx: (tensor) -- representing the indices of batch data features with shape of (None,
                               field_size).
        """

```

```

:param val: (tensor) -- representing the values of batch data features with shape of (None, field_size).
:return: (tensor) -- representing the output tensor of linear module with shape of (None).
"""
return torch.mul(input=self.weights.forward(input=idx).squeeze(dim=-1), other=val).sum(dim=-1,
keepdim=False)

class GlobalBiasModule(nn.Module):
    """Define the global bias module of DeepFM model."""
    def __init__(self):
        """Initialize the class of bias module"""
        super(GlobalBiasModule, self).__init__()
        self.bias = nn.Parameter(data=torch.tensor(data=0.0, dtype=torch.float32, requires_grad=False),
                               requires_grad=True)

    def forward(self):
        """Standard forward of the module.

        :return: (tensor) -- representing the global bias with shape of None.
        """
        return self.bias

class FactorizationMachinesModule(nn.Module):
    """Define the factorization machines module of DeepFM model."""
    def __init__(self):
        """Initialize the class of factorization machines module."""
        super(FactorizationMachinesModule, self).__init__()

    def forward(self, embeds):
        """Standard forward of the module.

        :param embeds: (tensor) -- representing the tensor after embedding-lookup operation with shape of
                           (None, field_size, embedding_size)
        :return: (tensor) -- representing the output tensor of factorzation machines module with shape of
                           (None).
        """
        ps = embeds.sum(dim=1, keepdim=False).pow(exponent=2) # a tensor with shape of (None, embedding_size)
        qs = embeds.pow(exponent=2).sum(dim=1, keepdim=False) # a tensor with shape of (None, embedding_size)
        return torch.mul(input=torch.sub(input=ps, other=qs).sum(dim=-1, keepdim=False), other=0.5)

class MultilayerPerceptronModule(nn.Module):
    """Define the multilayer perceptron(deep) module of DeepFM model."""
    def __init__(self, field_size, embedding_size, deep_hidden_layer_size, use_deep_bias, use_bn, use_dropout):
        """Initialize the class of MultilayerPerceptronModule module.

        :param field_size: (int) -- representing the number of fields.
        :param embedding_size: (int) -- representing the dimension of embedding vector.
        :param deep_hidden_layer_size: (list) -- containing the number of units belong to all hidden layers.
        :param use_deep_bias: (bool) -- indicating whether to use bias in all deep hidden layers.
        :param use_bn: (bool) -- indicating whether to perform batch normalization in all deep hidden layers.
        """

```

```

:param use_dropout: (bool) -- indicating whether to perform dropout (with constant dropout rate 0.15) in
                           all deep hidden layers.

"""
super(MultilayerPerceptronModule, self).__init__()
sequence = []
in_features = field_size * embedding_size
for i in range(len(deep_hidden_layer_size)): # construct connections between input layer and first
                                             hidden layer,
    # also between any two hidden layers
    # update out_features
    out_features = deep_hidden_layer_size[i]

    # add dense connection layer
    sequence +=[nn.Linear(in_features=in_features, out_features=out_features, bias=use_deep_bias)]
    # add batch normalization layer
    if use_bn:
        sequence +=[nn.BatchNorm1d(num_features=out_features, affine=True, track_running_stats=True)]
    # add activation layer
    sequence +=[nn.LeakyReLU(negative_slope=0.2, inplace=True)]
    # add dropout layer
    if use_dropout:
        sequence +=[nn.Dropout(p=0.15)]

    # update in_features
    in_features = out_features
# construct connections between the last hidden layer and logit output layer
sequence +=[nn.Linear(in_features=deep_hidden_layer_size[-1], out_features=1, bias=False)]

self.net = nn.Sequential(*sequence)

def forward(self, embeds):
    """Standard forward of the module.

:param embeds: (tensor) -- representing the tensor after embedding-lookup operation with shape of
                           (None, field_size, embedding_size)
:return: (tensor) -- representing the output tensor of deep module with shape of (None).
"""
    assert isinstance(embeds, torch.Tensor)
    inputs = embeds.flatten(start_dim=1, end_dim=-1) # a tensor with shape of (None, field_size *
                                                   embedding_size)

    return self.net.forward(input=inputs).squeeze(dim=-1)

class DeepFMNetRegression(nn.Module):
    """Define the class of DeepFM network. Only support for regression task."""
    def __init__(self,
                 field_size,
                 feature_size,
                 embedding_size,
                 deep_hidden_layer_size,
                 use_linear,

```

```

        use_global_bias,
        use_deep_bias,
        use_bn,
        use_dropout):
    """Initialize the class of DeepFM network for regression task.

:param field_size: (int) -- representing the number of fields.
:param feature_size: (int) -- representing the number of features(sum of numerical features and
                           category features).
:param embedding_size: (int) -- representing the dimension of embedding vector.
:param deep_hidden_layer_size: (list) -- containing the number of units belong to all hidden layers.
:param use_linear: (bool) -- indicating whether to construct linear module in deepfm network.
:param use_global_bias: (bool) -- indicating whether to construct global bias module in deepfm network.
:param use_deep_bias: (bool) -- indicating whether to use bias in all deep hidden layers.
:param use_bn: (bool) -- indicating whether to perform batch normalization in all deep hidden layers.
:param use_dropout: (bool) -- indicating whether to perform dropout(with constant dropout rate 0.15) in
                           all deep hidden layers.

"""
super(DeepFMNetRegression, self).__init__()
self.use_linear =use_linear
self.use_global_bias =use_global_bias

self.embeddings =EmbeddingModule(feature_size=feature_size, embedding_size=embedding_size)
self.fm =FactorizationMachinesModule()
self.mlp =MultilayerPerceptronModule(field_size=field_size,
                                      embedding_size=embedding_size,
                                      deep_hidden_layer_size=deep_hidden_layer_size,
                                      use_deep_bias=use_deep_bias,
                                      use_bn=use_bn,
                                      use_dropout=use_dropout)

if self.use_linear:
    self.linear =LinearModule(feature_size=feature_size)
if self.use_global_bias:
    self.gb =GlobalBiasModule()

def forward(self, idx, val):
    """Standard forward of the module.

:param idx: (tensor) -- representing the feature indices of batch data with shape of (None, field_size).
:param val: (tensor) -- representing the feature values of batch data with shape of (None, field_size).
:return outputs: (tensor) -- representing the output values of batch data with shape of (None)
"""

# embedding lookup operation
embeds =self.embeddings(idx=idx, val=val) # a tensor with shape of (None, field_size, embedding_size)

# interaction(factorization machines) part of the model
logits_fm =self.fm(embeds=embeds) # a tensor with shape of (None)

# deep(multilayer perceptron) part of the model
logits_deep =self.mlp(embeds=embeds) # a tensor with shape of (None)

outputs =torch.add(input=logits_fm, other=logits_deep) # a tensor with shape of (None)

```

```

# linear part of the model
if self.use_linear:
    logits_lr =self.linear(idx=idx, val=val) # a tensor with shape of (None)
    outputs =torch.add(input=outputs, other=logits_lr) # a tensor with shape of (None)

# global part of the model
if self.use_global_bias:
    logits_gb =self.gb() # a tensor with shape of None (but if add with a tensor with shape of (None),
                           the shape of result tensor will be (None))
    outputs =torch.add(input=outputs, other=logits_gb) # a tensor with shape of (None)

return outputs

class DeepFMNetBinary(nn.Module):
    """Define the class of DeepFM network. Only support for binary task."""
    def __init__(self,
                 field_size,
                 feature_size,
                 embedding_size,
                 deep_hidden_layer_size,
                 use_linear,
                 use_global_bias,
                 use_deep_bias,
                 use_bn,
                 use_dropout):
        """Initialize the class of DeepFM network for binary task.

        :param field_size: (int) -- representing the number of fields.
        :param feature_size: (int) -- representing the number of features(sum of numerical features and
                                      category features).
        :param embedding_size: (int) -- representing the dimension of embedding vector.
        :param deep_hidden_layer_size: (list) -- containing the number of units belong to all hidden layers.
        :param use_linear: (bool) -- indicating whether to construct linear module in deepfm network.
        :param use_global_bias: (bool) -- indicating whether to construct global bias module in deepfm network.
        :param use_deep_bias: (bool) -- indicating whether to use bias in all deep hidden layers.
        :param use_bn: (bool) -- indicating whether to perform batch normalization in all deep hidden layers.
        :param use_dropout: (bool) -- indicating whether to perform dropout(with constant dropout rate 0.15) in
                               all deep hidden layers.
        """
        super(DeepFMNetBinary, self).__init__()
        self.use_linear =use_linear
        self.use_global_bias =use_global_bias

        self.embeddings =EmbeddingModule(feature_size=feature_size, embedding_size=embedding_size)
        self.fm =FactorizationMachinesModule()
        self.mlp =MultilayerPerceptronModule(field_size=field_size,
                                              embedding_size=embedding_size,
                                              deep_hidden_layer_size=deep_hidden_layer_size,
                                              use_deep_bias=use_deep_bias,
                                              use_bn=use_bn,

```

```

        use_dropout=use_dropout)

    if self.use_linear:
        self.linear =LinearModule(feature_size=feature_size)
    if self.use_global_bias:
        self.gb =GlobalBiasModule()
    self.sigmoid =nn.Sigmoid() # a layer transforming logit output into probability output

def forward(self, idx, val):
    """Standard forward of the module.

    :param idx: (tensor) -- representing the feature indices of batch data with shape of (None, field_size).
    :param val: (tensor) -- representing the feature values of batch data with shape of (None, field_size).
    :return outputs: (tensor) -- representing the output values of batch data with shape of (None)
    """

    # embedding lookup operation
    embeds =self.embeddings(idx=idx, val=val) # a tensor with shape of (None, field_size, embedding_size)

    # interaction(factorization machines) part of the model
    logits_fm =self.fm(embeds=embeds) # a tensor with shape of (None)

    # deep(multilayer perceptron) part of the model
    logits_deep =self.mlp(embeds=embeds) # a tensor with shape of (None)

    logits =torch.add(input=logits_fm, other=logits_deep) # a tensor with shape of (None)

    # linear part of the model
    if self.use_linear:
        logits_lr =self.linear(idx=idx, val=val) # a tensor with shape of (None)
        logits =torch.add(input=logits, other=logits_lr) # a tensor with shape of (None)

    # global part of the model
    if self.use_global_bias:
        logits_gb =self.gb() # a tensor with shape of None (but if add with a tensor with shape of (None),
                             # the shape of result tensor will be (None))
        logits =torch.add(input=logits, other=logits_gb) # a tensor with shape of (None)
    outputs =self.sigmoid.forward(input=logits) # a tensor with shape of (None)

    return outputs

def init_parameters(net, init_type="xavier"):
    """Initialize network parameters.

    :param net: (torch.nn.Module) -- representing the network of model.
    :param init_type: (str, optional) -- indicating the type of initialization for weights.
                           Valid value in ["normal", "uniform", "xavier", "kaiming", "orthogonal"].
                           (default: "xavier")
    """

    def init_func(m): # define the initialization function
        if isinstance(m, nn.Embedding): # initialize for nn.Embedding module(variables of module only include
                                       # weight)
            # initialize for weight of nn.Embedding module

```

```

    if init_type == "normal":
        nn.init.normal_(tensor=m.weight, mean=0.0, std=1.0)
    elif init_type == "uniform":
        nn.init.uniform_(tensor=m.weight, a=-1.0, b=1.0)
    elif init_type == "xavier":
        nn.init.xavier_normal_(tensor=m.weight, gain=1.0)
    elif init_type == "kaiming":
        nn.init.kaiming_normal_(tensor=m.weight, a=0, mode="fan_in")
    elif init_type == "orthogonal":
        nn.init.orthogonal_(tensor=m.weight, gain=1.0)
    else:
        raise NotImplementedError("initialization method [%s] is not implemented" % init_type)

    elif isinstance(m, nn.Linear): # initialize for nn.Linear module(variables of module include weight and bias)
        # initialize for weight of nn.Linear module
        if init_type == "normal":
            nn.init.normal_(tensor=m.weight, mean=0.0, std=1.0)
        elif init_type == "uniform":
            nn.init.uniform_(tensor=m.weight, a=-1.0, b=1.0)
        elif init_type == "xavier":
            nn.init.xavier_normal_(tensor=m.weight, gain=1.0)
        elif init_type == "kaiming":
            nn.init.kaiming_normal_(tensor=m.weight, a=0, mode="fan_in")
        elif init_type == "orthogonal":
            nn.init.orthogonal_(tensor=m.weight, gain=1.0)
        else:
            raise NotImplementedError("initialization method [%s] is not implemented" % init_type)
        # initialize for bias of nn.Linear module
        if m.bias is not None:
            nn.init.zeros_(m.bias)

    elif isinstance(m, nn.BatchNorm1d): # initialize for nn.BatchNorm1d module(variables of module include weight and bias)
        # initialize for weight of nn.BatchNorm1d module
        nn.init.constant_(tensor=m.weight, val=1.0)
        # initialize for bias of nn.BatchNorm1d module
        nn.init.constant_(tensor=m.bias, val=0.0)

        # apply the initialization function <init_func>
        net.apply(fn=init_func)

def init_network(net, device_ids=[], init_type="xavier"):
    """Initialize the network, including initialization of network weights and network topology (support cpu, single-gpu and multi-gpus mode).

    :param net: (torch.nn.Module) -- representing the network of model.
    :param device_ids: (list, optional) -- containing all gpu devices, and indicating the type of network topology.
        if len(device_ids) == 0: cpu mode;
        if len(device_ids) == 1: single gpu mode;
    """

```

```

        if len(device_ids) > 1: multi gpu mode.
        (default: [] (cpu mode))

:param init_type: (str, optional) -- indicating the type of initialization for weights.
                    Valid value in ["normal", "uniform", "xavier", "kaiming", "orthogonal"].
                    (default: "xavier")

:return: (torch.nn.Module) -- representing the network after initialization of network weights and network
                                topology.

"""

if len(device_ids) ==0: # cpu mode
    if torch.cuda.is_available():
        print("[GPU] is currently available to be used but use [CPU] now")
    init_parameters(net=net, init_type=init_type)
    return net

elif len(device_ids) ==1: # single-gpu mode
    device =torch.device(device="cuda:%d" % device_ids[0] if torch.cuda.is_available() else "cpu")
    if device.type == "cpu":
        print("GPU [cuda:%d] is currently not available to be used, we use [CPU] instead" % device_ids[0])
    net.to(device=device)
    init_parameters(net=net, init_type=init_type)
    return net

else: # multi-gpus mode
    device =torch.device(device="cuda:%d" % device_ids[0] if torch.cuda.is_available() else "cpu")
    if device.type == "cpu":
        gpu_setting_str =""
        for idx in device_ids:
            gpu_setting_str += "%d," % idx
        gpu_setting_str =gpu_setting_str.rstrip(",")
        print("GPU [cuda:%s] is currently not available to be used, we use [CPU] instead" % gpu_setting_str)
    net.to(device=device)
    init_parameters(net=net, init_type=init_type)
    return net

else:
    if torch.cuda.device_count() >1:
        network =nn.DataParallel(module=net, device_ids=device_ids, output_device=device_ids[0], dim=0)
        network.to(device=device)
        init_parameters(net=network, init_type=init_type)
        return network
    else:
        print("GPU available number is less than expected number [%d], we use [CPU] instead" %
              len(device_ids))
        init_parameters(net=net, init_type=init_type)
        return net

def fetch_deepfm_network(params):
    """Fetches the object of torch.nn.Module class representing the network of DeepFM.

:param params: (object) -- containing all hyper-parameters of model.
:return: (torch.nn.Module) -- representing the network of model.

"""

```

```

# Declare all hyper-parameters of model controled by params
task_type =params.task_type
field_size =params.field_size
feature_size =params.feature_size
embedding_size =params.embedding_size
deep_hidden_layer_size =params.deep_hidden_layer_size
use_deep_bias =params.use_deep_bias
use_global_bias =params.use_global_bias
use_linear =params.use_linear
use_bn =params.use_bn
use_dropout =params.use_dropout
device_ids =params.device_ids
init_type =params.init_type

if task_type == "regression":
    # Define the network structure
    net =DeepFMNetRegression(field_size=field_size,
                             feature_size=feature_size,
                             embedding_size=embedding_size,
                             deep_hidden_layer_size=deep_hidden_layer_size,
                             use_deep_bias=use_deep_bias,
                             use_global_bias=use_global_bias,
                             use_linear=use_linear,
                             use_bn=use_bn,
                             use_dropout=use_dropout)

    # Initialize the network topology and all network parameters
    network =init_network(net=net, device_ids=device_ids, init_type=init_type)
    return network

elif task_type == "binary":
    # Define the network structure
    net =DeepFMNetBinary(field_size=field_size,
                          feature_size=feature_size,
                          embedding_size=embedding_size,
                          deep_hidden_layer_size=deep_hidden_layer_size,
                          use_deep_bias=use_deep_bias,
                          use_global_bias=use_global_bias,
                          use_linear=use_linear,
                          use_bn=use_bn,
                          use_dropout=use_dropout)

    # Initialize the network topology and all network parameters
    network =init_network(net=net, device_ids=device_ids, init_type=init_type)
    return network

else:
    raise NotImplementedError("task_type of model [%s] is not supported" % task_type)

```

objective

```

# Copyright (C) 2019 Tong Jia. All rights reserved.
import torch.nn as nn

def fetch_loss_fn(task_type):

```

```

"""Fetch loss function based on specific task type.

:param task_type: (str) -- indicating the type of task taken value in ["binary", "regression"].
:return loss_fn: (nn.BCELoss or MSELoss()) -- representing the loss function.
"""

if task_type == "binary":
    loss_fn = nn.BCELoss()
elif task_type == "regression":
    loss_fn = nn.MSELoss()
else:
    raise NotImplementedError("task type [%s] is not supported, it must be in [\"binary\",
                                \"regression\"]")

return loss_fn


def compute_regularization(net, lambda_reg, reg_type):
    """Compute the regularization loss of the model.

    :param net: (torch.nn.Module) -- representing the network of model.
    :param lambda_reg: (float) -- representing the coefficient of regularization term.
    :return (tensor) -- representing the regularization loss of model with shape of None(scalar).
    """

    if reg_type == "l1":
        p = 1
    elif reg_type == "l2":
        p = 2
    else:
        raise NotImplementedError("reg_type [%s] is not supported, it must be in [\"l1\", \"l2\"]" % reg_type)

    MODULE_LIST = ["linear"] # (optional hyper-parameter), another choice is MODULE_LIST = ["linear", "mlp"]
    reg = 0
    for param in net.named_parameters():
        param_name = param[0].strip().split(".")
        if param_name[0] in MODULE_LIST and param_name[-1] == "weight":
            reg += param[1].norm(p=p, keepdim=False)
    return reg * lambda_reg

```

optimizer

```

# Copyright (C) 2019 Tong Jia. All rights reserved.
from torch.optim.sgd import SGD
from torch.optim.adam import Adam
import torch.nn as nn


def fetch_optimizer(optimizer_type, net, lr):
    assert isinstance(net, nn.Module)
    if optimizer_type == "sgd":
        optimizer = SGD(params=net.parameters(), lr=lr)
    elif optimizer_type == "adam":
        optimizer = Adam(params=net.parameters(), lr=lr, betas=(0.9, 0.999))

```

```

    else:
        raise NotImplementedError("optimizer_type [%s] is not supported." % optimizer_type)

    return optimizer

```

metrics

```

# Copyright (C) 2019 Tong Jia. All rights reserved.

import numpy as np
from sklearn.metrics import recall_score, accuracy_score, roc_auc_score
from sklearn.metrics import mean_squared_error

# -----Define evaluation metircs for binary classification task-----

def compute_recall(outputs, targets):
    """Compute recall score for current batch data.

    :param outputs: (np.ndarray) -- representing the batch outputs with shape of (None).
    :param targets: (np.ndarray) -- representing the batch targets with shape of (None).
    :return: (float) -- representing the recall score.
    """
    return recall_score(y_true=targets, y_pred=np.round(a=outputs, decimals=0))

def compute_accuracy(outputs, targets):
    """Compute accuracy score for current batch data.

    :param outputs: (np.ndarray) -- representing the batch outputs with shape of (None).
    :param targets: (np.ndarray) -- representing the batch targets with shape of (None).
    :return: (float) -- representing the accuracy score.
    """
    return accuracy_score(y_true=targets, y_pred=np.round(a=outputs, decimals=0))

def compute_auc(outputs, targets):
    """Compute auc score for current batch data.

    :param outputs: (np.ndarray) -- representing the batch outputs with shape of (None).
    :param targets: (np.ndarray) -- representing the batch targets with shape of (None).
    :return: (float) -- representing the auc score.
    """
    auc_score = roc_auc_score(y_true=targets, y_score=np.round(a=outputs, decimals=0))
    return auc_score

# -----Define evaluation metircs for regression task-----
def compute_mse(outputs, targets):
    """Compute mean squared error(mse) for current batch data.

    :param outputs: (np.ndarray) -- representing the batch outputs with shape of (None).
    :param targets: (np.ndarray) -- representing the batch targets with shape of (None).
    :return: (float) -- representing the mse score.
    """

```

```

    return mean_squared_error(y_true=targets, y_pred=outputs)

def fetch_evaluation_metrics(task_type):
    """Fetch evaluation metrics dict based on specific task type.
    In practical computation usage, we use the function as follow:
    metrics_dict = fetch_evaluation_metrics(task_type="binary")
    summary_batch = {metric_name: metrics_dict[metric_name](outputs=batch_outputs, targets=batch_targets)
                    for metric_name in metrics_dict.keys()}
    where batch_outputs and batch_targets are both torch tensor with the same shape as [None],
    and the values in dict summary_batch are corresponding numpy numerical scalars.

:param task_type: (str) -- indicating the type of task taken value in ["binary", "regression"].
:return metrics: (dict) -- containing correspond evaluation metrics functions.
"""

if task_type == "binary":
    metrics = {
        "recall": compute_recall,
        "accuracy": compute_accuracy,
        "auc": compute_auc
        # could add more metrics later
    }
    return metrics
elif task_type == "regression":
    metrics = {
        "mse": compute_mse
        # could add more metrics later
    }
    return metrics
else:
    raise NotImplementedError("task type [%s] is not supported, it must be in [%s, %s]" %
                             ("binary", "regression"))

```

```

utils
# Copyright (C) 2019 Tong Jia. All rights reserved.

import os
import json
import shutil
import torch
import codecs
import numpy as np

class RunningAverage(object):
    """A simple class that maintains the running average of a quantity

Example:
```
loss_avg = RunningAverage()
loss_avg.update(2)
loss_avg.update(4)
loss_avg() = 3
```

```

```

```
"""

def __init__(self):
 self.__steps = 0
 self.__total = 0

 def update(self, val):
 self.__total +=val
 self.__steps +=1

 def __call__(self):
 return self.__total /float(self.__steps)

def save_checkpoint(state, is_best, checkpoint_dir):
 """Save network structure and parameters at checkpoint_dir + "last.pth.tar", if argument is_best == True,
 also save
 checkpoint_dir + "best.pth.tar".

 :param state: (dict) -- containing network's state_dict, may containing other keys and corresponding values
 such as
 epoch, optimizer's state_dict and so on.
 :param is_best: (bool) -- indicating whether it is the best model seen till now.
 :param checkpoint_dir: (str) -- indicating the folder where parameters to be saved.
 """
 filepath =os.path.join(checkpoint_dir, "last.pth.tar")
 if not os.path.exists(path=filepath):
 print("checkpoint directory does not exist! Making directory {}".format(checkpoint_dir))
 os.mkdir(path=checkpoint_dir)
 else:
 print("checkpoint directory exist")
 torch.save(obj=state, f=filepath)
 if is_best:
 shutil.copyfile(src=filepath, dst=os.path.join(checkpoint_dir, "best.pth.tar"))

def load_checkpoint(checkpoint, network, optimizer=None):
 """Load model structure and parameters from checkpoint, if optimizer exist, load optimizer hyper-parameters.

 :param checkpoint: (str) -- indicating the file path which containing parameters information.
 :param network: (torch.nn.Module) -- representing the network for which the parameters are loaded.
 :param optimizer: (torch.optim, optional) -- resuming optimizer from checkpoint_dir.
 :return: (dict) -- containing network's state_dict, may containing other keys and corresponding values such
 as
 epoch, optimizer's state_dict and so on.
 """
 if not os.path.exists(path=checkpoint):
 raise ValueError("File [{}] doesn't exist!".format(checkpoint))
 ckpt =torch.load(f=checkpoint)
 network.load_state_dict(state_dict=ckpt["network_state_dict"])
 if optimizer is not None:
 optimizer.load_state_dict(state_dict=ckpt["optimizer_state_dict"])

```

```
return ckpt

def save_dict_to_json(d, json_path):
 """Save dict of float into json file.

 :param d: (dict) -- containing float-castable values (np.float, int, float, etc.)
 :param json_path: (str) -- representing the path to json file.
 """
 with codecs.open(filename=json_path, mode="w") as f:
 d = {k: np.float(v) for k, v in d.items()}
 json.dump(obj=d, fp=json_path, indent=4) # ???

def set_logger(log_path):
 raise NotImplementedError
```

## 54.7 xDeepFM

### 54.7.1 Introduction

本节来自论文：xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems[?]

### 54.7.2 Model formulation

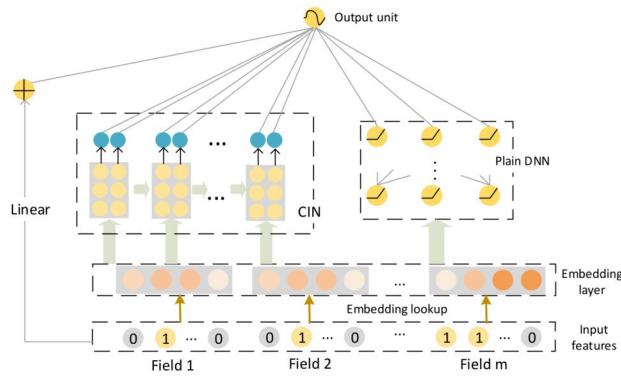


图 179: The structure of xDeepFM[?]

### Model inference

## 54.8 Deep Interest Network (DIN)

### 54.8.1 Introduction

论文出处：Deep Interest Network for Click-Through Rate Prediction[?]

### 54.8.2 Model formulation

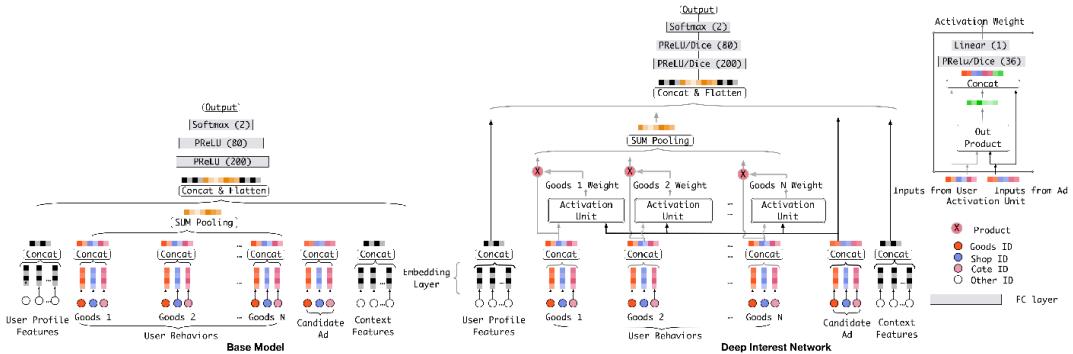


图 180: The structure of Deep Interest Network[?]

### Model inference

### 54.8.3 Model implement

```
"""
Created on 2019/03/25 02:52
author: Tong Jia
email: cecilio.jia@gmail.com
description:
 An implementation of Deep Interest Network for Click-Through Rate Prediction (DIN).
 See algorithm and hyperparameter details:
 [Zhou et al., 2018] (https://arxiv.org/pdf/1706.06978.pdf)
 The algorithm is developed with TensorFlow Estimator based on TensorFlow 1.12.0 version.
"""

import tensorflow as tf
import tensorflow.contrib as tfc
import os

FLAGS = tf.flags.FLAGS
-----Hyperparameters of estimator-----
tf.flags.DEFINE_enum(name="phase",
 default=None,
 enum_values=["train", "eval", "predict", "export"],
 help="A string, representing the phase of estimator")
tf.flags.DEFINE_string(name="model_dir",
 default=None,
 help="A string, representing the folder path of saved model files")
tf.flags.DEFINE_string(name="export_dir",
```

```

 default=None,
 help="A string, representing the basic folder path of export .pb files")
-----Hyperparameters of estimator (optional)-----
tf.flags.DEFINE_boolean(name="perform_valid_during_train",
 default=True,
 help="(optional) A boolean, instructing whether to perform validation on valid dataset
 during train phase")
tf.flags.DEFINE_integer(name="log_step_count_steps",
 default=500,
 help="(optional) An integer, representing the frequency, in number of global steps, that
 the global step/sec will be
 logged during training",
 lower_bound=1,
 upper_bound=None)
tf.flags.DEFINE_integer(name="save_checkpoints_steps",
 default=20000,
 help="(optional) An integer, representing save checkpoints every this many steps",
 lower_bound=1,
 upper_bound=None)
tf.flags.DEFINE_integer(name="keep_checkpoint_max",
 default=2,
 help="(optional) An integer, representing max number of saved model checkpoints",
 lower_bound=1,
 upper_bound=None)
-----Hyperparameters of input function-----
tf.flags.DEFINE_string(name="data_dir",
 default=None,
 help="A string, representing the folder path of dataset")
tf.flags.DEFINE_string(name="delimiter",
 default=None,
 help="A string, separating consecutive <index j>:<value j> pairs in a line of data file")
tf.flags.DEFINE_string(name="separator",
 default=None,
 help="A string, separating feature index(left part) and feature value(right part) in a
 specific pair")
tf.flags.DEFINE_integer(name="batch_size",
 default=None,
 help="An integer, representing the number of consecutive elements of this dataset to
 combine in a single batch",
 lower_bound=1,
 upper_bound=None)
tf.flags.DEFINE_integer(name="epochs",
 default=None,
 help="An integer, representing the number of times the dataset should be repeated",
 lower_bound=1,
 upper_bound=None)
tf.flags.DEFINE_integer(name="field_size_user_profile",
 default=None,
 help="An integer scalar, representing the number of fields belong to user profile",
 lower_bound=1,
 upper_bound=None)
tf.flags.DEFINE_integer(name="field_size_item_profile",

```

```

 default=None,
 help="An integer scalar, representing the number of fields belong to item profile",
 lower_bound=1,
 upper_bound=None)
tf.flags.DEFINE_integer(name="field_size_context",
 default=None,
 help="An integer scalar, representing the number of fields belong to context",
 lower_bound=1,
 upper_bound=None)
tf.flags.DEFINE_integer(name="behaviors_size",
 default=None,
 help="An integer scalar, representing the length of user behaviors list",
 lower_bound=1,
 upper_bound=None)
-----Hyperparameters of input function (optional)-----
tf.flags.DEFINE_boolean(name="shuffle",
 default=True,
 help="(optional) A boolean, instructing whether to randomly shuffle the samples of
 training dataset")

tf.flags.DEFINE_integer(name="buffer_size",
 default=100000,
 help="(optional) An integer scalar, denoting the number of bytes to buffer",
 lower_bound=1,
 upper_bound=None)
tf.flags.DEFINE_integer(name="num_parallel_calls",
 default=4,
 help="(optional) An integer scalar, representing the number elements to process in
 parallel",
 lower_bound=1,
 upper_bound=None)
tf.flags.DEFINE_boolean(name="use_dtype_high_precision",
 default=False,
 help="(optional) A boolean, instructing the dtype of both input function and model
 function. If False, use
 tf.float32; if True, use
 tf.float64")

tf.flags.DEFINE_string(name="name_feat_inds_user",
 default="user_profile_inds",
 help="(optional) A string, representing the name of feature indices of user profile fields
 in return dict")

tf.flags.DEFINE_string(name="name_feat_vals_user",
 default="user_profile_vals",
 help="(optional) A string, representing the name of feature values of user profile fields
 in return dict")

tf.flags.DEFINE_string(name="name_feat_inds_item",
 default="item_profile_inds",
 help="(optional) A string, representing the name of feature indices of item profile fields
 in return dict")

tf.flags.DEFINE_string(name="name_feat_vals_item",
 default="item_profile_vals",
 help="(optional) A string, representing the name of feature values of item profile fields
 in return dict")

```

```

tf.flags.DEFINE_string(name="name_feat_inds_context",
 default="cont_inds",
 help="(optional) A string, representing the name of feature indices of context fields in
 return dict")

tf.flags.DEFINE_string(name="name_feat_vals_context",
 default="cont_vals",
 help="(optional) A string, representing the name of feature values of context fields in
 return dict")

tf.flags.DEFINE_string(name="name_feat_inds_candidate",
 default="cand_inds",
 help="(optional) A string, representing the name of feature index of candidate in return
 dict")

tf.flags.DEFINE_string(name="name_feat_inds_behaviors",
 default="beha_inds",
 help="(optional) A string, representing the name of feature indices of user behaviors in
 return dict")

-----Hyperparameters of model function-----
tf.flags.DEFINE_string(name="task",
 default=None,
 help="A string, representing the type of task (binary or regression)")

tf.flags.DEFINE_integer(name="output_size",
 default=None,
 help="An integer scalar, representing the number of output units",
 lower_bound=1,
 upper_bound=None)

tf.flags.DEFINE_integer(name="feat_size_user_profile",
 default=None,
 help="An integer scalar, representing the number of features belong to user profile",
 lower_bound=1,
 upper_bound=None)

tf.flags.DEFINE_integer(name="feat_size_item_profile",
 default=None,
 help="An integer scalar, representing the number of features belong to item profile",
 lower_bound=1,
 upper_bound=None)

tf.flags.DEFINE_integer(name="feat_size_context",
 default=None,
 help="An integer scalar, representing the number of features belong to context",
 lower_bound=1,
 upper_bound=None)

tf.flags.DEFINE_integer(name="feat_size_id",
 default=None,
 help="An integer scalar, representing the number of ids of item(for candidate and user
 behaviors)",
 lower_bound=1,
 upper_bound=None)

tf.flags.DEFINE_integer(name="embed_size_user_profile",
 default=None,
 help="An integer scalar, representing the dimension of embedding vectors for all features
 belong to user profile fields",
 lower_bound=1,
 upper_bound=None)

```

```

tf.flags.DEFINE_integer(name="embed_size_item_profile",
 default=None,
 help="An integer scalar, representing the dimension of embedding vectors for all features
 belong to item profile fields",
 lower_bound=1,
 upper_bound=None)
tf.flags.DEFINE_integer(name="embed_size_context",
 default=None,
 help="An integer scalar, representing the dimension of embedding vectors for all features
 belong to context fields",
 lower_bound=1,
 upper_bound=None)
tf.flags.DEFINE_integer(name="embed_size_id",
 default=None,
 help="An integer scalar, representing the dimension of embedding vectors for candidate
 and all behaviors (item IDs)",
 lower_bound=1,
 upper_bound=None)
tf.flags.DEFINE_list(name="hidden_sizes",
 default=None,
 help="A list, containing the number of hidden units of each hidden layer in dnn part (e.g.
 128,64,32,16)")
tf.flags.DEFINE_list(name="dropouts",
 default=None,
 help="A list or None, containing the dropout rate of each hidden layer in dnn part "
 "(e.g. 0.4,0.3,0.2,0.1); if None, don't use dropout operation for any hidden layer")
-----Hyperparameters of model function (optional)-----
tf.flags.DEFINE_boolean(name="use_softmax_norm_for_attention",
 default=False,
 help="A boolean, instructing whether to perform normalization with softmax on the output
 of attention scores between
 candidate embedding and user
 behaviors embeddings")
tf.flags.DEFINE_boolean(name="use_bn",
 default=True,
 help="(optional) A boolean, instructing whether to use batch normalization for each
 hidden layer in dnn part")
tf.flags.DEFINE_boolean(name="use_global_bias",
 default=True,
 help="(optional) A boolean, instructing whether to use global bias in model inference")
tf.flags.DEFINE_boolean(name="use_hidden_bias",
 default=True,
 help="(optional) A boolean, instructing whether to use bias of hidden layer units in
 model inference")
tf.flags.DEFINE_float(name="lamb",
 default=0.001,
 help="(optional) A float scalar, representing the coefficient of regularization term",
 lower_bound=0.0,
 upper_bound=None)
tf.flags.DEFINE_string(name="optimizer",
 default="adam",
 help="(optional) A string, representing the type of optimizer")

```

```

tf.flags.DEFINE_float(name="learning_rate",
 default=0.003,
 help="(optional) A float scalar, representing the learning rate of optimizer",
 lower_bound=0.0,
 upper_bound=None)

def input_fn(filenames,
 delimiter,
 separator,
 batch_size,
 epochs,
 field_size_user_profile,
 field_size_item_profile,
 field_size_context,
 behaviors_size,
 shuffle=True,
 buffer_size=100000,
 num_parallel_calls=4,
 dtype=tf.float32,
 name_feat_inds_user="user_profile_inds",
 name_feat_vals_user="user_profile_vals",
 name_feat_inds_item="item_profile_inds",
 name_feat_vals_item="item_profile_vals",
 name_feat_inds_context="cont_inds",
 name_feat_vals_context="cont_vals",
 name_feat_inds_candidate ="cand_inds",
 name_feat_inds_behaviors ="beha_inds"):

 """
 The input function for loading sparse dataset in Deep Interest Network format. The columns are separated by
 argument
 delimiter(e.g. " ").
 Note:
 1. From left to right in each line, there contains six parts in order:
 label => user fields => item fields => context fields => candidate index => user behaviors indices
 <label>
 <index 1 user>:<value 1 user> ... <index j_u user>:<value j_u user>
 <index 1 item>:<value 1 item> ... <index j_i item>:<value j_i item>
 <index 1 context>:<value 1 context> ... <index j_c context>:<value j_c context>
 <index candidate>
 <index 1 behaviors> ... <index j_bmax behaviors>
 2. Each fields group maintains an index-system independently, namely:
 a. There exist index 1 in both user fields group and item fields group
 b. The meaning of the first-index value in user fields group is different from the first-index value
 in item fields group
 3. The feature type of each fields group:
 user fields(fixed length > 1): dense field, one-hot active field
 item fields(fixed length > 1): dense field, one-hot active field
 context fields(fixed length > 1): dense field, one-hot active field
 candidate field(fixed length = 1): one-hot active field
 user behaviors indices(fixed length = behaviors_size): multi-hot active field
 4. The order of behaviors indices is unnecessary because of sum pooling operation later.
 5. The input function can be used for binary classification, multi classification and regression task:
 """

```

```

 binary classification: <label> in {0, 1};
 multi classification: <label> in {0, K} (K is the number of total classes);
 regression: <label> in (-inf, inf).

e.g.
(field_size_user_profile = 2, field_size_item_profile = 3, field_size_context = 1, behaviors_size = 5)
1 0:1 6:1 0:1 10:1 17:0.456127 0:1 14 1 4 5 8 9
0 0:1 9:1 3:1 7:1 17:0.077712 3:1 2 7 3 6 18 21 88 83 13 22 71
if the length of user behaviors is greater than <behaviors_size>, we will select first <behaviors_size>
element from left to right.

For the first sample:
1: label
0:1 6:1: features of user profile
0:1 10:1 17:0.456127: features of item profile
0:1: features of context
14: index of candidate
1 4 5 8 9: indices of user behaviors

Parameters

:param filenames: list
 A list of string, containing one or more paths of filenames.
:param delimiter: str
 A str, separating consecutive <index j>:<value j> pairs in data files.
:param separator: str
 A str, separating feature index(left part of key-value pair) and feature value(right part of key-value
 pair) in one specific pair.
:param batch_size: int
 An integer scalar, representing the number of consecutive elements of this dataset to combine in a
 single batch.
:param epochs: int
 An integer scalar, representing the number of times the dataset should be repeated.
:param field_size_user_profile: int
 An integer scalar, representing the number of fields(number of columns before one-hot encoding) of user
 profile.
:param field_size_item_profile: int
 An integer scalar, representing the number of fields(number of columns before one-hot encoding) of item
 profile.
:param field_size_context: int
 An integer scalar, representing the number of fields(number of columns before one-hot encoding) of
 context.
:param behaviors_size: int
 An integer scalar, representing the length of user behaviors list.
:param shuffle: boolean, optional
 A boolean(defaults to True), instructing whether to randomly shuffle the elements of this dataset.
:param buffer_size: int, optional
 An integer scalar(defaults to 100000), denoting the number of bytes to buffer.
:param num_parallel_calls: int, optional
 An integer scalar(defaults to 4), representing the number elements to process in parallel.
:param dtype: tf.Dtype, optional
 A tf.DType(defaults to tf.float32), representing the numeric type of values. it always takes value from
 [tf.float32, tf.float64].
:param name_feat_inds_user: str, optional

```

```

 A string, representing the name of feature indices of user profile fields in return dict.
:param name_feat_vals_user: str, optional
 A string, representing the name of feature values of user profile fields in return dict.
:param name_feat_inds_item: str, optional
 A string, representing the name of feature indices of item profile fields in return dict.
:param name_feat_vals_item: str, optional
 A string, representing the name of feature values of item profile fields in return dict.
:param name_feat_inds_context: str, optional
 A string, representing the name of feature indices of context fields in return dict.
:param name_feat_vals_context: str, optional
 A string, representing the name of feature values of context fields in return dict.
:param name_feat_inds_candidate: str, optional
 A string, representing the name of feature index of candidate in return dict.
:param name_feat_inds_behaviors: str, optional
 A string, representing the name of feature indices of user behaviors in return dict.

>Returns

:return: dict
 A dict of eight Tensors, representing features(including feature indices and feature values) in a
single batch.

{
 <name_feat_inds_user>: tf.Tensor of feature indices of user profile fields in shape of (None,
 field_size_user_profile),
 <name_feat_vals_user>: tf.Tensor of feature values of user profile fields in shape of (None,
 field_size_user_profile),
 <name_feat_inds_item>: tf.Tensor of feature indices of item profile fields in shape of (None,
 field_size_item_profile),
 <name_feat_vals_item>: tf.Tensor of feature values of item profile fields in shape of (None,
 field_size_item_profile),
 <name_feat_inds_context>: tf.Tensor of feature indices of context fields in shape of (None,
 field_size_context),
 <name_feat_vals_context>: tf.Tensor of feature values of context in shape of (None,
 field_size_context)
 <name_feat_inds_candidate>: tf.Tensor of feature index of candidate in shape of (None)
 <name_feat_inds_behaviors>: tf.Tensor of feature indices of user behaviors in shape of (None,
 behaviors_size)
}

:return Tensor
 A Tensor in shape of (None), representing labels in a single batch.

"""
START_COLUMN_FEATS_USER =1
END_COLUMN_FEATS_USER =START_COLUMN_FEATS_USER +field_size_user_profile
START_COLUMN_FEATS_ITEM =END_COLUMN_FEATS_USER
END_COLUMN_FEATS_ITEM =START_COLUMN_FEATS_ITEM +field_size_item_profile
START_COLUMN_FEATS_CONT =END_COLUMN_FEATS_ITEM
END_COLUMN_FEATS_CONT =START_COLUMN_FEATS_CONT +field_size_context
START_COLUMN_FEAT_CANDIDATE =END_COLUMN_FEATS_CONT
START_COLUMN_FEAT_USER_BEHAVIORS =START_COLUMN_FEAT_CANDIDATE +1
END_COLUMN_FEAT_USER_BEHAVIORS =START_COLUMN_FEAT_USER_BEHAVIORS +behaviors_size
def map_func(line):
 columns =tf.string_split(source=[line], delimiter=delimiter, skip_empty=False).values

```

```

-----Process label-----
label =tf.string_to_number(string_tensor=columns[0], out_type=dtype)

-----Process user profile fields-----
splits_user =tf.string_split(source=columns[START_COLUMN_FEATS_USER: END_COLUMN_FEATS_USER],
 delimiter=separator,
 skip_empty=False)
feats_user =tf.reshape(tensor=splits_user.values, shape=splits_user.dense_shape)
inds_user, vals_user =tf.split(value=feats_user, num_or_size_splits=2, axis=1) # Two tensors in shape
 of (field_size_user_profile, 1)
feat_inds_user =tf.squeeze(input=tf.string_to_number(string_tensor=inds_user, out_type=tf.int32),
 axis=1) # A tensor in shape of
 (field_size_user_profile)
feat_vals_user =tf.squeeze(input=tf.string_to_number(string_tensor=vals_user, out_type=dtype), axis=1)
 # A tensor in shape of (field_size_user_profile)
-----Process item profile fields (not contain item index)-----
splits_item =tf.string_split(source=columns[START_COLUMN_FEATS_ITEM: END_COLUMN_FEATS_ITEM],
 delimiter=separator,
 skip_empty=False)
feats_item =tf.reshape(tensor=splits_item.values, shape=splits_item.dense_shape)
inds_item, vals_item =tf.split(value=feats_item, num_or_size_splits=2, axis=1)
feat_inds_item =tf.squeeze(input=tf.string_to_number(string_tensor=inds_item, out_type=tf.int32),
 axis=1) # A tensor in shape of
 (field_size_item_profile)
feat_vals_item =tf.squeeze(input=tf.string_to_number(string_tensor=vals_item, out_type=dtype), axis=1)
 # A tensor in shape of (field_size_item_profile)
-----Process context fields-----
splits_cont =tf.string_split(source=columns[START_COLUMN_FEATS_CONT: END_COLUMN_FEATS_CONT],
 delimiter=separator,
 skip_empty=False)
feats_cont =tf.reshape(tensor=splits_cont.values, shape=splits_cont.dense_shape)
inds_cont, vals_cont =tf.split(value=feats_cont, num_or_size_splits=2, axis=1)
feat_inds_cont =tf.squeeze(input=tf.string_to_number(string_tensor=inds_cont, out_type=tf.int32),
 axis=1) # A tensor in shape of
 (field_size_context)
feat_vals_cont =tf.squeeze(input=tf.string_to_number(string_tensor=vals_cont, out_type=dtype), axis=1)
 # A tensor in shape of (field_size_context)
-----Process candidate item index-----
feat_inds_cand =tf.string_to_number(string_tensor=columns[START_COLUMN_FEAT_CANDIDATE],
 out_type=tf.int32)

-----Process user behaviors (associated item indices)-----
feat_inds beha =tf.string_to_number(string_tensor=columns[START_COLUMN_FEAT_USER_BEHAVIORS:
 END_COLUMN_FEAT_USER_BEHAVIORS],
 out_type=tf.int32)

feats =[

 name_feat_inds_user: feat_inds_user,
 name_feat_vals_user: feat_vals_user,
 name_feat_inds_item: feat_inds_item,
 name_feat_vals_item: feat_vals_item,
 name_feat_inds_context: feat_inds_cont,
 name_feat_vals_context: feat_vals_cont,
 name_feat_inds_candidate: feat_inds_cand,
]

```

```

 name_feat_inds_behaviors: feat_inds_bela
 }
 return feats, label

dataset =tf.data.TextLineDataset(filenames=filenames, buffer_size=buffer_size). \
 map(map_func=map_func, num_parallel_calls=num_parallel_calls)
if (shuffle ==True):
 dataset =dataset.shuffle(buffer_size=buffer_size)
dataset =dataset. \
 repeat(count=epochs). \
 batch(batch_size=batch_size, drop_remainder=False)
dataset =dataset.prefetch(buffer_size=1)
iterator =dataset.make_one_shot_iterator()
batch_feats, batch_labels =iterator.get_next()
return batch_feats, batch_labels

def model_fn(features, labels, mode, params):
 """Model function of Deep interest network(DIN) for predictive analytics of high dimensional sparse data.

 Args of dict params:
 task: str
 A string, representing the type of task.
 Note:
 it must take value from ["binary", "multi", "regression"];
 it instruct the type of loss function:
 "binary": sigmoid cross-entropy;
 "multi": softmax cross-entropy;
 "regression": mean squared error.
 output_size: int
 An integer scalar, representing the number of output units.
 Note:
 it must be correspond to <task>:
 task == "binary": output_size must be equal to 1;
 task == "multi": output_size must be equal to the dimension of class distribution;
 task == "regression": output_size must be equal to 1.
 field_size_user_profile: int
 An integer scalar, representing the number of fields(number of columns before one-hot encoding) of
 user profile.
 Note:
 it must be consistent with <field_size_user_profile> of input function.
 field_size_item_profile: int
 An integer scalar, representing the number of fields(number of columns before one-hot encoding) of
 item profile.
 Note:
 it must be consistent with <field_size_item_profile> of input function.
 field_size_context: int
 An integer scalar, representing the number of fields(number of columns before one-hot encoding) of
 context.
 Note:
 it must be consistent with <field_size_context> of input function.
 feat_size_user_profile: int
 """

```

An integer scalar, representing the number of features(number of columns after one-hot encoding) of user profile.

`feat_size_item_profile: int`  
An integer scalar, representing the number of features(number of columns after one-hot encoding) of item profile.

`feat_size_context: int`  
An integer scalar, representing the number of features(number of columns after one-hot encoding) of context.

`feat_size_id: int`  
An integer scalar, representing the number of features(number of id after one-hot encoding) of item(for candidate and user behaviors).

`embed_size_user_profile: int`  
An integer scalar, representing the dimension of embedding vectors for all features belong to user profile fields.

`embed_size_item_profile: int`  
An integer scalar, representing the dimension of embedding vectors for all features belong to item profile fields.

`embed_size_context: int`  
An integer scalar, representing the dimension of embedding vectors for all features belong to context fields.

`embed_size_id: int`  
An integer scalar, representing the dimension of embedding vectors for candidate and all behaviors (item IDs).

`hidden_sizes: list`  
A list, containing the number of hidden units of each hidden layer in dnn part.  
Note:  
it doesn't contain output layer of dnn part.

`dropouts: list or None`  
If list, containing the dropout rate of each hidden layer in dnn part;  
If None, don't use dropout operation for any hidden layer.  
Note:  
if list, the length of <dropouts> must be equal to <hidden\_sizes>.

`use_softmax_norm_for_attention: bool`  
A boolean, instructing whether to perform normalization with softmax on the output of attention scores between candidate embedding and user behaviors embeddings.  
Note:  
In raw paper, it don't suggest to perform normalization with softmax on the output of attention scores.  
For example, if one user's historical behaviors contain 90% clothes and 10% electronics. Given two candidate ads of T-shirt and phone, T-shirt activates most of the historical behaviors belonging to clothes and may get larger value of  $v_U$  (higher intensity of interest) than phone. Traditional methods lose the resolution on the numerical scale of  $v_U$  by normalizing of the output of  $a(\cdot)$ . So:  
if all behaviors and all candidates are almost similar(come from the same category), we can set  
`use_softmax_norm_for_attention == True;`  
else, we can `use_softmax_norm_for_attention == False` to get better representation ability of intensity of interest.

```

use_bn: bool
 A boolean, instructing whether to use batch normalization for each hidden layer in dnn part.
use_hidden_bias: bool
 A boolean, instructing whether to use bias of hidden layer units in model inference.
use_global_bias: bool
 A boolean, instructing whether to use global bias in model inference.
lamb: float
 A float scalar, representing the coefficient of regularization term (the larger the value of lamb,
 the stronger the penalty is).

Note:
 Here regularization is only used for global bias.

optimizer: str
 A string, representing the type of optimizer.

learning_rate: float
 A float scalar, representing the learning rate of optimizer.

dtype: tf.Dtype
 A tf.DType, representing the numeric type of values.

Note:
 it must take value from [tf.float32, tf.float64];
 it must be consistent with <dtype> of input function.

name_feat_inds_user: str, optional
 A string, representing the name of feature indices of user profile fields in return dict.

Note:
 it must be consistent with <name_feat_inds_user> of input function.

name_feat_vals_user: str, optional
 A string, representing the name of feature values of user profile fields in return dict.

Note:
 it must be consistent with <name_feat_vals_user> of input function.

name_feat_inds_item: str, optional
 A string, representing the name of feature indices of item profile fields in return dict.

Note:
 it must be consistent with <name_feat_inds_item> of input function.

name_feat_vals_item: str, optional
 A string, representing the name of feature values of item profile fields in return dict.

Note:
 it must be consistent with <name_feat_vals_item> of input function.

name_feat_inds_context: str, optional
 A string, representing the name of feature indices of context fields in return dict.

Note:
 it must be consistent with <name_feat_inds_context> of input function.

name_feat_vals_context: str, optional
 A string, representing the name of feature values of context fields in return dict.

Note:
 it must be consistent with <name_feat_vals_context> of input function.

name_feat_inds_candidate: str, optional
 A string, representing the name of feature index of candidate in return dict.

Note:
 it must be consistent with <name_feat_inds_candidate> of input function.

name_feat_inds_behaviors: str, optional
 A string, representing the name of feature indices of user behaviors in return dict.

Note:
 it must be consistent with <name_feat_inds_behaviors> of input function.

```

```

reuse: bool
 A boolean, which takes value from [False, True, tf.AUTO_REUSE].
seed: int or None
 If integer scalar, representing the random seed of tensorflow;
 If None, random choice.

"""
-----Declare all hyperparameters from params-----
task =params["task"]
output_size =params["output_size"]
field_size_user_profile =params["field_size_user_profile"]
field_size_item_profile =params["field_size_item_profile"]
field_size_context =params["field_size_context"]
feat_size_user_profile =params["feat_size_user_profile"]
feat_size_item_profile =params["feat_size_item_profile"]
feat_size_context =params["feat_size_context"]
feat_size_id =params["feat_size_id"]
embed_size_user_profile =params["embed_size_user_profile"]
embed_size_item_profile =params["embed_size_item_profile"]
embed_size_context =params["embed_size_context"]
embed_size_id =params["embed_size_id"]
hidden_sizes =params["hidden_sizes"]
dropouts =params["dropouts"]
use_softmax_norm_for_attention =params["use_softmax_norm_for_attention"]
use_bn =params["use_bn"]
use_hidden_bias =params["use_hidden_bias"]
use_global_bias =params["use_global_bias"]
lamb =params["lamb"]
optimizer =params["optimizer"]
learning_rate =params["learning_rate"]
dtype =params["dtype"]
name_feat_inds_user =params["name_feat_inds_user"]
name_feat_vals_user =params["name_feat_vals_user"]
name_feat_inds_item =params["name_feat_inds_item"]
name_feat_vals_item =params["name_feat_vals_item"]
name_feat_inds_context =params["name_feat_inds_context"]
name_feat_vals_context =params["name_feat_vals_context"]
name_feat_inds_candidate =params["name_feat_inds_candidate"]
name_feat_inds_behaviors =params["name_feat_inds_behaviors"]
reuse =params["reuse"]
seed =params["seed"]
----Hyperparameters for threshold for binary classification task
threshold =0.5
-----Hyperparameters for exponential decay(*manual optional*)-----
decay_steps =5000
decay_rate =0.998
staircase =True
----Hyperparameters for information showing-----
name_probability_output ="prob"
name_classification_output ="class"
name_regression_output ="pred"
value_error_warning_task ="Argument of model function <task>: \"{}\" is not supported. It must be in " \
"[\"binary\", \"multi\", \"regression\"]".format(task)

```

```

value_error_warning_optimizer ="Argument value of <optimizer>: {} is not supported.".format(optimizer)
value_error_warning_output_size_and_task ="Argument of model function <output_size>: {}, must be 1 when
 <task> " \
 "is: \\"{}\"".format(output_size, task)

-----Assert for hyperparameters-----
if task == "binary":
 assert (output_size == 1), value_error_warning_output_size_and_task
if task == "regression":
 assert (output_size == 1), value_error_warning_output_size_and_task

if seed !=None:
 tf.set_random_seed(seed=seed)

-----Build model inference-----
with tf.variable_scope(name_or_scope="inference", reuse=reuse):
 with tf.name_scope(name="inputs"):
 ids_u =features[name_feat_inds_user] # A tensor in shape of (None, field_size_user_profile)
 x_u =features[name_feat_vals_user] # A tensor in shape of (None, field_size_user_profile)
 ids_i =features[name_feat_inds_item] # A tensor in shape of (None, field_size_item_profile)
 x_i =features[name_feat_vals_item] # A tensor in shape of (None, field_size_item_profile)
 ids_co =features[name_feat_inds_context]
 x_co =features[name_feat_vals_context]
 ids_ca =features[name_feat_inds_candidate]
 ids_be =features[name_feat_inds_behaviors]

 with tf.name_scope(name="embed-user-profile-layer"):
 Vu =tf.get_variable(name="Vu",
 shape=[feat_size_user_profile, embed_size_user_profile],
 dtype=dtype,
 initializer=tfc.layers.xavier_initializer(uniform=False, dtype=dtype),
 regularizer=None)
 vu =tf.nn.embedding_lookup(params=Vu, ids=ids_u) # A tensor in shape of (None,
 field_size_user_profile,
 embed_size_user_profile)

 with tf.name_scope(name="embed-item-profile-layer"):
 Vi =tf.get_variable(name="Vi",
 shape=[feat_size_item_profile, embed_size_item_profile],
 dtype=dtype,
 initializer=tfc.layers.xavier_initializer(uniform=False, dtype=dtype),
 regularizer=None)
 vi =tf.nn.embedding_lookup(params=Vi, ids=ids_i) # A tensor in shape of (None,
 field_size_item_profile,
 embed_size_item_profile)

 with tf.name_scope(name="embed-context-layer"):
 Vc =tf.get_variable(name="Vc",
 shape=[feat_size_context, embed_size_context],
 dtype=dtype,
 initializer=tfc.layers.xavier_initializer(uniform=False, dtype=dtype),
 regularizer=None)

```

```

vc =tf.nn.embedding_lookup(params=Vc, ids=ids_co) # A tensor in shape of (None, field_size_context,
 embed_size_context)

with tf.name_scope(name="embed-id-layer"):
 Vid =tf.get_variable(name="Vid",
 shape=[feat_size_id, embed_size_id],
 dtype=dtype,
 initializer=tfc.layers.xavier_initializer(uniform=False, dtype=dtype),
 regularizer=None)
 # -----embedding look up operation for candidate-----
 vca =tf.nn.embedding_lookup(params=Vid, ids=ids_ca) # A tensor in shape of (None, embed_size_id)
 # -----embedding look up operation for user behaviors-----
 vbe =tf.nn.embedding_lookup(params=Vid, ids=ids_be) # A tensor in shape of (None, behaviors_size,
 embed_size_id)

with tf.name_scope(name="attention-id-layer"):
 # -----Key part of DIN model-----
 vca_reshape =tf.expand_dims(input=vca, axis=1) # A tensor in shape of (None, 1, embed_size_id)
 # We can use many methods to build up attention function such as dot-products, feed-forward network
 # and so on, here we use dot-product method
 attention =tf.reduce_sum(input_tensor=tf.multiply(x=vca_reshape, y=vbe), # A tensor in shape of
 (None, behaviors_size, embed_size_id)
 axis=-1,
 keepdims=False) # A tensor in shape of (None, behaviors_size)
 if use_softmax_norm_for_attention ==True:
 attention =tf.nn.softmax(logits=attention, axis=-1)
 attention =tf.expand_dims(input=attention, axis=-1) # A tensor in shape of (None, behaviors_size, 1)
 vembed =tf.reduce_sum(input_tensor=tf.multiply(x=attention, y=vbe), # A tensor in shape of (None,
 behaviors_size, embed_size_id)
 axis=1,
 keepdims=False) # A tensor in shape of (None, embed_size_id)

with tf.name_scope(name="concat-flatten-layer"):
 x_u_reshape =tf.expand_dims(input=x_u, axis=-1) # A tensor in shape of (None,
 field_size_user_profile, 1)
 x_i_reshape =tf.expand_dims(input=x_i, axis=-1) # A tensor in shape of (None,
 field_size_item_profile, 1)
 x_co_reshape =tf.expand_dims(input=x_co, axis=-1) # A tensor in shape of (None, field_size_context,
 1)
 eu =tf.multiply(x=x_u_reshape, y=vu) # A tensor in shape of (None, field_size_user_profile,
 embed_size_user_profile)
 ei =tf.multiply(x=x_i_reshape, y=vi) # A tensor in shape of (None, field_size_item_profile,
 embed_size_item_profile)
 ec =tf.multiply(x=x_co_reshape, y=vc) # A tensor in shape of (None, field_size_context,
 embed_size_context)
 eu_reshape =tf.reshape(tensor=eu, shape=[-1, field_size_user_profile *embed_size_user_profile])
 ei_reshape =tf.reshape(tensor=ei, shape=[-1, field_size_item_profile *embed_size_item_profile])
 ec_rehsape =tf.reshape(tensor=ec, shape=[-1, field_size_context *embed_size_context])
 logits =tf.concat(values=[eu_reshape, ei_reshape, ec_rehsape, vca, vembed], axis=1) # A tensor in
 shape of (None, dnn_input_size)

with tf.name_scope(name="dnn-hidden-layer"):

```

```

for l in range(len(hidden_sizes)):
 # -----The order for each hidden layer is: matmul => bn => relu => dropout => matmul => ...
 logits =tf.layers.dense(inputs=logits,
 units=hidden_sizes[l],
 activation=None,
 use_bias=use_hidden_bias,
 kernel_initializer=tfc.layers.xavier_initializer(uniform=True, dtype=dtype),
 bias_initializer=tf.zeros_initializer(dtype=dtype),
 kernel_regularizer=None, # *manual optional*
 bias_regularizer=None, # *manual optional*
 name="mlp-dense-hidden-{}".format(l))

 if use_bn ==True:
 logits =tf.layers.batch_normalization(inputs=logits,
 axis=-1,
 momentum=0.99, # *manual optional*
 epsilon=1e-3, # *manual optional*
 center=True,
 scale=True,
 beta_initializer=tf.zeros_initializer(dtype=dtype),
 gamma_initializer=tf.ones_initializer(dtype=dtype),
 moving_mean_initializer=tf.zeros_initializer(dtype=dtype),
 moving_variance_initializer=tf.ones_initializer(dtype=dtype),
 beta_regularizer=None, # *manual optional*
 gamma_regularizer=None, # *manual optional*
 training=(mode ==tf.estimator.ModeKeys.TRAIN),
 name="mlp-bn-hidden-{}".format(l))

 logits =tf.nn.relu(features=logits)
 if dropouts !=None:
 logits =tf.layers.dropout(inputs=logits,
 rate=dropouts[l],
 seed=seed,
 training=(mode ==tf.estimator.ModeKeys.TRAIN))

 with tf.name_scope(name="output"):
 logits =tf.layers.dense(inputs=logits,
 units=output_size,
 activation=None,
 use_bias=use_global_bias,
 kernel_initializer=tfc.layers.xavier_initializer(uniform=True, dtype=dtype),
 bias_initializer=tf.zeros_initializer(dtype=dtype),
 kernel_regularizer=None, # *manual optional*
 bias_regularizer=tfc.layers.l2_regularizer(scale=lamb), # *manual optional*
 name="mlp-dense-output") # A tensor in shape of (None, output_size)

 if task =="binary":
 logits =tf.squeeze(input=logits, axis=1) # A tensor in shape of (None)
 probs =tf.nn.sigmoid(x=logits)
 classes =tf.cast(x=tf.greater(x=tf.nn.sigmoid(x=logits), y=threshold), dtype=tf.int32)
 predictions ={
 name_probability_output: probs,
 name_classification_output: classes
 }

```

```

 elif task == "multi":
 probs_dist = tf.nn.softmax(logits=logits, axis=-1)
 classes = tf.argmax(input=logits, axis=-1, output_type=tf.int32)
 predictions = {
 name_probability_output: probs_dist,
 name_classification_output: classes
 }
 elif task == "regression":
 logits = tf.squeeze(input=logits, axis=1) # A tensor in shape of (None)
 predictions = {
 name_regression_output: logits
 }
 else:
 raise ValueError(value_error_warning_task)

-----Provide an estimator spec for `ModeKeys.PREDICTION` mode-----
if mode == tf.estimator.ModeKeys.PREDICT:
 export_outputs = {
 tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
 tf.estimator.export.PredictOutput(outputs=predictions)
 } # For usage of tensorflow serving
 return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions, export_outputs=export_outputs)

-----Build loss function-----
if task == "binary":
 loss = tf.reduce_mean(input_tensor=tf.nn.sigmoid_cross_entropy_with_logits(labels=labels,
 logits=logits),
 axis=0,
 keepdims=False) # A scalar, representing the training loss of current batch training
 dataset
elif task == "regression":
 loss = tf.reduce_mean(input_tensor=tf.square(x=tf.subtract(x=labels, y=logits)),
 axis=0,
 keepdims=False) # A scalar, representing the training loss of current batch training
 dataset
elif task == "multi":
 labels_one_hot = tf.one_hot(indices=tf.cast(x=labels, dtype=tf.int32),
 depth=output_size,
 axis=-1,
 dtype=dtype) # A tensor in shape of (None, output_size)
 loss = tf.reduce_mean(input_tensor=tf.nn.softmax_cross_entropy_with_logits_v2(labels=labels_one_hot,
 logits=logits),
 axis=0,
 keepdims=False) # A scalar, representing the training loss of current batch training
 dataset
else:
 raise ValueError(value_error_warning_task)

reg = tf.reduce_sum(input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
 axis=0,
 keepdims=False,
 name="regularization") # A scalar, representing the regularization loss of current batch

```

```

 training dataset

loss += reg

-----Provide an estimator spec for `ModeKeys.EVAL` mode-----
if mode == tf.estimator.ModeKeys.EVAL:
 if task == "binary":
 eval_metric_ops = {
 "accuracy": tf.metrics.accuracy(labels=labels,
 predictions=predictions[name_classification_output]),
 "precision": tf.metrics.precision(labels=labels,
 predictions=predictions[name_classification_output]),
 "recall": tf.metrics.recall(labels=labels, predictions=predictions[name_classification_output]),
 "auc": tf.metrics.auc(labels=labels, predictions=predictions[name_classification_output])
 }
 elif task == "multi":
 eval_metric_ops = {
 "confusion-matrix": tf.confusion_matrix(labels=labels,
 predictions=predictions[name_classification_output],
 num_classes=output_size)
 }
 elif task == "regression":
 eval_metric_ops = {
 "rmse": tf.metrics.root_mean_squared_error(labels=labels,
 predictions=predictions[name_regression_output])
 }
 else:
 raise ValueError(value_error_warning_task)
return tf.estimator.EstimatorSpec(mode=mode, loss=loss, predictions=predictions,
 eval_metric_ops=eval_metric_ops)

-----Build optimizer-----
global_step = tf.train.get_or_create_global_step(graph=tf.get_default_graph()) # Define a global step for
 training step counter
if optimizer == "sgd":
 opt_op = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
elif optimizer == "sgd-exp-decay":
 decay_learning_rate = tf.train.exponential_decay(learning_rate=learning_rate,
 global_step=global_step,
 decay_steps=decay_steps,
 decay_rate=decay_rate,
 staircase=staircase)
 opt_op = tf.train.GradientDescentOptimizer(learning_rate=decay_learning_rate)
elif optimizer == "momentum":
 opt_op = tf.train.MomentumOptimizer(learning_rate=learning_rate,
 momentum=0.9,
 use_nesterov=False)
elif optimizer == "momentum-exp-decay":
 decay_learning_rate = tf.train.exponential_decay(learning_rate=learning_rate,
 global_step=global_step,
 decay_steps=decay_steps,
 decay_rate=decay_rate,
 staircase=staircase)

```

```

opt_op =tf.train.MomentumOptimizer(learning_rate=decay_learning_rate,
 momentum=0.9,
 use_nesterov=False)
elif optimizer == "nesterov":
 opt_op =tf.train.MomentumOptimizer(learning_rate=learning_rate,
 momentum=0.9,
 use_nesterov=True)
elif optimizer == "nesterov-exp-decay":
 decay_learning_rate =tf.train.exponential_decay(learning_rate=learning_rate,
 global_step=global_step,
 decay_steps=decay_steps,
 decay_rate=decay_rate,
 staircase=staircase)
opt_op =tf.train.MomentumOptimizer(learning_rate=decay_learning_rate,
 momentum=0.9,
 use_nesterov=True)
elif optimizer == "adagrad":
 opt_op =tf.train.AdagradOptimizer(learning_rate=learning_rate,
 initial_accumulator_value=0.1)
elif optimizer == "adadelta":
 opt_op =tf.train.AdadeltaOptimizer(learning_rate=learning_rate,
 rho=0.95)
elif optimizer == "rmsprop":
 opt_op =tf.train.RMSPropOptimizer(learning_rate=learning_rate,
 decay=0.9)
elif optimizer == "adam":
 opt_op =tf.train.AdamOptimizer(learning_rate=learning_rate,
 beta1=0.9,
 beta2=0.999)
else:
 raise NotImplementedError(value_error_warning_optimizer)

train_op =opt_op.minimize(loss=loss, global_step=global_step, name="train_op")

-----Provide an estimator spec for `ModeKeys.TRAIN` mode-----
if (mode ==tf.estimator.ModeKeys.TRAIN):
 return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

def main(unused_argv):
 # -----Declare all hyperparameters of terminal interface-----
 phase =FLAGS.phase
 model_dir =FLAGS.model_dir
 export_dir =FLAGS.export_dir
 perform_valid_during_train =FLAGS.perform_valid_during_train # (optional) Note: if use validation dataset
 # during train phase, the valid set and train set are
 # in same data_dir
 log_step_count_steps =FLAGS.log_step_count_steps # (optional)
 save_checkpoints_steps =FLAGS.save_checkpoints_steps # (optional)
 keep_checkpoint_max =FLAGS.keep_checkpoint_max # (optional)
 data_dir =FLAGS.data_dir
 delimiter =FLAGS.delimiter

```

```

separator =FLAGS.separator
batch_size =FLAGS.batch_size
epochs =FLAGS.epochs
field_size_user_profile =FLAGS.field_size_user_profile
field_size_item_profile =FLAGS.field_size_item_profile
field_size_context =FLAGS.field_size_context
behaviors_size =FLAGS.behaviors_size
shuffle =FLAGS.shuffle # (optional)
buffer_size =FLAGS.buffer_size # (optional)
num_parallel_calls =FLAGS.num_parallel_calls # (optional)
use_dtype_high_precision =FLAGS.use_dtype_high_precision # (optional)
name_feat_inds_user =FLAGS.name_feat_inds_user # (optional)
name_feat_vals_user =FLAGS.name_feat_vals_user # (optional)
name_feat_inds_item =FLAGS.name_feat_inds_item # (optional)
name_feat_vals_item =FLAGS.name_feat_vals_item # (optional)
name_feat_inds_context =FLAGS.name_feat_inds_context # (optional)
name_feat_vals_context =FLAGS.name_feat_vals_context # (optional)
name_feat_inds_candidate =FLAGS.name_feat_inds_candidate # (optional)
name_feat_inds_behaviors =FLAGS.name_feat_inds_behaviors # (optional)
task =FLAGS.task
output_size =FLAGS.output_size
feat_size_user_profile =FLAGS.feat_size_user_profile
feat_size_item_profile =FLAGS.feat_size_item_profile
feat_size_context =FLAGS.feat_size_context
feat_size_id =FLAGS.feat_size_id
embed_size_user_profile =FLAGS.embed_size_user_profile
embed_size_item_profile =FLAGS.embed_size_item_profile
embed_size_context =FLAGS.embed_size_context
embed_size_id =FLAGS.embed_size_id
hidden_sizes =FLAGS.hidden_sizes
dropouts =FLAGS.dropouts
use_softmax_norm_for_attention =FLAGS.use_softmax_norm_for_attention # (optional)
use_bn =FLAGS.use_bn # (optional)
use_global_bias =FLAGS.use_global_bias # (optional)
use_hidden_bias =FLAGS.use_hidden_bias # (optional)
lamb =FLAGS.lamb # (optional)
optimizer =FLAGS.optimizer # (optional)
learning_rate =FLAGS.learning_rate # (optional)

PREFIX_TRAIN_FILE ="train"
PREFIX_EVAL_FILE ="eval"
PREFIX_PREDICT_FILE ="predict"
REUSE =False
SEED =None

if use_dtype_high_precision ==False:
 dtype =tf.float32
else:
 dtype =tf.float64
hidden_sizes =[int(float(ele)) for ele in hidden_sizes]
if dropouts !=None:
 dropouts =[float(ele) for ele in dropouts]

```

```

hparams ={
 "task": task,
 "output_size": output_size,
 "field_size_user_profile": field_size_user_profile,
 "field_size_item_profile": field_size_item_profile,
 "field_size_context": field_size_context,
 "feat_size_user_profile": feat_size_user_profile,
 "feat_size_item_profile": feat_size_item_profile,
 "feat_size_context": feat_size_context,
 "feat_size_id": feat_size_id,
 "embed_size_user_profile": embed_size_user_profile,
 "embed_size_item_profile": embed_size_item_profile,
 "embed_size_context": embed_size_context,
 "embed_size_id": embed_size_id,
 "hidden_sizes": hidden_sizes,
 "dropouts": dropouts,
 "use_softmax_norm_for_attention": use_softmax_norm_for_attention,
 "use_bn": use_bn,
 "use_global_bias": use_global_bias,
 "use_hidden_bias": use_hidden_bias,
 "lamb": lamb,
 "optimizer": optimizer,
 "learning_rate": learning_rate,
 "dtype": dtype,
 "name_feat_inds_user": name_feat_inds_user,
 "name_feat_vals_user": name_feat_vals_user,
 "name_feat_inds_item": name_feat_inds_item,
 "name_feat_vals_item": name_feat_vals_item,
 "name_feat_inds_context": name_feat_inds_context,
 "name_feat_vals_context": name_feat_vals_context,
 "name_feat_inds_candidate": name_feat_inds_candidate,
 "name_feat_inds_behaviors": name_feat_inds_behaviors,
 "reuse": REUSE,
 "seed": SEED
}

-----Multi-GPU Usage-----
mirrored_strategy = tfc.distribute.MirroredStrategy(devices=["/gpu:0", "/gpu:1"])
config = tf.estimator.RunConfig(
log_step_count_steps=log_step_count_steps,
save_checkpoints_steps=save_checkpoints_steps,
keep_checkpoint_max=keep_checkpoint_max,
train_distribute=mirrored_strategy,
eval_distribute=mirrored_strategy
)

config =tf.estimator.RunConfig(
 log_step_count_steps=log_step_count_steps,
 save_checkpoints_steps=save_checkpoints_steps,
 keep_checkpoint_max=keep_checkpoint_max
)

```

```

estimator =tf.estimator.Estimator(model_fn=model_fn, model_dir=model_dir, params=hparams, config=config)
if phase == "train":
 filenames_train =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_TRAIN_FILE + "*"))
 if perform_valid_during_train ==True:
 filenames_valid =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_EVAL_FILE + "*"))
 train_spec =tf.estimator.TrainSpec(input_fn=lambda :input_fn(filenames=filenames_train,
 delimiter=delimiter,
 separator=separator,
 batch_size=batch_size,
 epochs=epochs,
 field_size_user_profile=field_size_user_profile,
 field_size_item_profile=field_size_item_profile,
 field_size_context=field_size_context,
 behaviors_size=behaviors_size,
 shuffle=shuffle,
 buffer_size=buffer_size,
 num_parallel_calls=num_parallel_calls,
 dtype=dtype,
 name_feat_inds_user=name_feat_inds_user,
 name_feat_vals_user=name_feat_vals_user,
 name_feat_inds_item=name_feat_inds_item,
 name_feat_vals_item=name_feat_vals_item,
 name_feat_inds_context=name_feat_inds_context,
 name_feat_vals_context=name_feat_vals_context,
 name_feat_inds_candidate=name_feat_inds_candidate,
 name_feat_inds_behaviors=name_feat_inds_behaviors),
 max_steps=None)
 eval_spec =tf.estimator.EvalSpec(input_fn=lambda :input_fn(filenames=filenames_valid,
 delimiter=delimiter,
 separator=separator,
 batch_size=batch_size,
 epochs=1,
 field_size_user_profile=field_size_user_profile,
 field_size_item_profile=field_size_item_profile,
 field_size_context=feat_size_context,
 behaviors_size=behaviors_size,
 shuffle=False,
 buffer_size=buffer_size,
 num_parallel_calls=num_parallel_calls,
 dtype=dtype,
 name_feat_inds_user=name_feat_inds_user,
 name_feat_vals_user=name_feat_vals_user,
 name_feat_inds_item=name_feat_inds_item,
 name_feat_vals_item=name_feat_vals_item,
 name_feat_inds_context=name_feat_inds_context,
 name_feat_vals_context=name_feat_vals_context,
 name_feat_inds_candidate=name_feat_inds_candidate,
 name_feat_inds_behaviors=name_feat_inds_behaviors),
 steps=None,
 start_delay_secs=120,
 throttle_secs=600)
 tf.estimator.train_and_evaluate(estimator=estimator, train_spec=train_spec, eval_spec=eval_spec)

```

```

else:
 estimator.train(input_fn=lambda :input_fn(filenames=filenames_train,
 delimiter=delimiter,
 separator=separator,
 batch_size=batch_size,
 epochs=epochs,
 field_size_user_profile=field_size_user_profile,
 field_size_item_profile=field_size_item_profile,
 field_size_context=field_size_context,
 behaviors_size=behaviors_size,
 shuffle=shuffle,
 buffer_size=buffer_size,
 num_parallel_calls=num_parallel_calls,
 dtype=dtype,
 name_feat_inds_user=name_feat_inds_user,
 name_feat_vals_user=name_feat_vals_user,
 name_feat_inds_item=name_feat_inds_item,
 name_feat_vals_item=name_feat_vals_item,
 name_feat_inds_context=name_feat_inds_context,
 name_feat_vals_context=name_feat_vals_context,
 name_feat_inds_candidate=name_feat_inds_candidate,
 name_feat_inds_behaviors=name_feat_inds_behaviors),
 steps=None,
 max_steps=None)
elif phase == "eval":
 filenames_eval =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_EVAL_FILE +"*"))
 estimator.evaluate(input_fn=lambda :input_fn(filenames=filenames_eval,
 delimiter=delimiter,
 separator=separator,
 batch_size=batch_size,
 epochs=1,
 field_size_user_profile=field_size_user_profile,
 field_size_item_profile=field_size_item_profile,
 field_size_context=feat_size_context,
 behaviors_size=behaviors_size,
 shuffle=False,
 buffer_size=buffer_size,
 num_parallel_calls=num_parallel_calls,
 dtype=dtype,
 name_feat_inds_user=name_feat_inds_user,
 name_feat_vals_user=name_feat_vals_user,
 name_feat_inds_item=name_feat_inds_item,
 name_feat_vals_item=name_feat_vals_item,
 name_feat_inds_context=name_feat_inds_context,
 name_feat_vals_context=name_feat_vals_context,
 name_feat_inds_candidate=name_feat_inds_candidate,
 name_feat_inds_behaviors=name_feat_inds_behaviors))
elif phase == "predict":
 filenames_predict =tf.gfile.Glob(filename=os.path.join(data_dir, PREFIX_PREDICT_FILE +"*"))
 p = estimator.predict(input_fn=lambda :input_fn(filenames=filenames_predict,
 delimiter=delimiter,
 separator=separator,

```

```

 batch_size=batch_size,
 epochs=1,
 field_size_user_profile=field_size_user_profile,
 field_size_item_profile=field_size_item_profile,
 field_size_context=feat_size_context,
 behaviors_size=behaviors_size,
 shuffle=False,
 buffer_size=buffer_size,
 num_parallel_calls=num_parallel_calls,
 dtype=dtype,
 name_feat_inds_user=name_feat_inds_user,
 name_feat_vals_user=name_feat_vals_user,
 name_feat_inds_item=name_feat_inds_item,
 name_feat_vals_item=name_feat_vals_item,
 name_feat_inds_context=name_feat_inds_context,
 name_feat_vals_context=name_feat_vals_context,
 name_feat_inds_candidate=name_feat_inds_candidate,
 name_feat_inds_behaviors=name_feat_inds_behaviors))

-----Usage demo, still need to be accomplished-----
for ele in p:
 print(ele)
elif phase == "export":
 features ={
 name_feat_inds_user: tf.placeholder(dtype=tf.int32, shape=[None, field_size_user_profile],
 name=name_feat_inds_user),
 name_feat_vals_user: tf.placeholder(dtype=dtype, shape=[None, field_size_user_profile],
 name=name_feat_vals_user),
 name_feat_inds_item: tf.placeholder(dtype=tf.int32, shape=[None, field_size_item_profile],
 name=name_feat_inds_item),
 name_feat_vals_item: tf.placeholder(dtype=dtype, shape=[None, field_size_item_profile],
 name=name_feat_vals_item),
 name_feat_inds_context: tf.placeholder(dtype=dtype, shape=[None, field_size_context],
 name=name_feat_inds_context),
 name_feat_vals_context: tf.placeholder(dtype=dtype, shape=[None, field_size_context],
 name=name_feat_vals_context),
 name_feat_inds_candidate: tf.placeholder(dtype=tf.int32, shape=[None],
 name=name_feat_inds_candidate),
 name_feat_inds_behaviors: tf.placeholder(dtype=tf.int32, shape=[None, behaviors_size],
 name=name_feat_inds_behaviors)
 }
 serving_input_receiver_fn =tf.estimator.export.build_raw_serving_input_receiver_fn(features=features)
 estimator.export_savedmodel(export_dir_base=export_dir,
 serving_input_receiver_fn=serving_input_receiver_fn)
else:
 raise NotImplementedError("Argument <phase> value: {} is not supported.".format(phase))

if __name__ == '__main__':
 tf.logging.set_verbosity(v=tf.logging.INFO)
 tf.app.run(main=main)

```

## 54.9 Deep Interest Evolution Network (DIEN)

### 54.9.1 Introduction

本节来自论文：Deep Interest Evolution Network for Click-Through Rate Prediction[?]

### 54.9.2 Model formulation

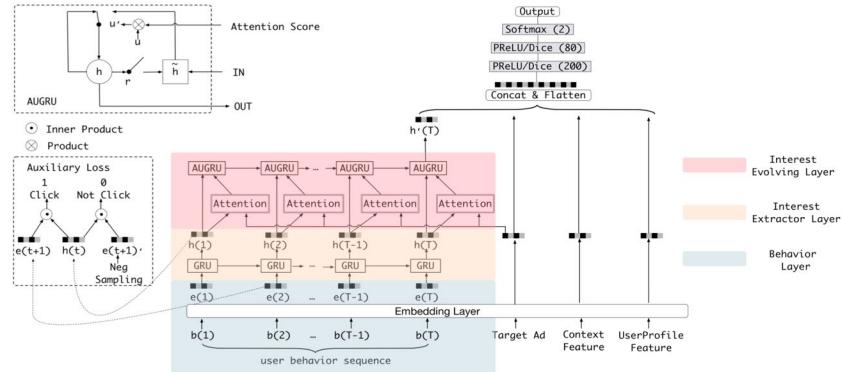


图 181: The structure of Deep Interest Evolution Network[?]

### Model inference

## 54.10 AutoInt

### 54.10.1 Introduction

本节来自论文：AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks[?]

### 54.10.2 Model formulation

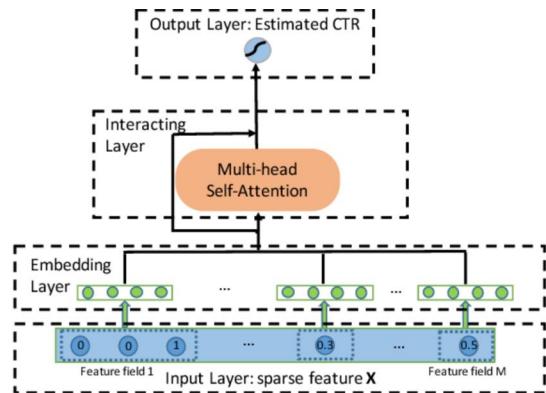


图 182: The structure of AutoInt[?]

### Model inference

#### 1. Input Layer

使用稀疏向量（sparse vector）表示 user 's profiles 以及 item 's attributes:

$$\mathbf{x} = [\mathbf{x}_1; \mathbf{x}_2; \dots; \mathbf{x}_M]$$

其中：

- $M$ : The number of feature fields
- $\mathbf{x}_i$ : The feature representation of the  $i$ -th field:
  - $\mathbf{x}_i$  is a one-hot vector if the  $i$ -th field is categorical
  - $\mathbf{x}_i$  is a scalar value if the  $i$ -th field is numerical

#### 2. Embedding Layer

由于 categorical features 的特征表示是高维稀疏的向量（high-dimensional one-hot vector），因此通常的做法是用低维空间（low-dimensional spaces）对其进行表示（e.g. word embeddings），因此：

$$\mathbf{e}_i = \mathbf{V}_i \mathbf{x}_i$$

其中：

- $\mathbf{V}_i$ : The embedding matrix of  $i$ -th feature field
- $\mathbf{x}_i$ : The one-hot vector of  $i$ -th feature field
- $\mathbf{e}_i \in \mathbb{R}^d$ : The embedding vector of  $i$ -th feature field

注意：为了允许实现 categorical feature 和 numerical feature 的交叉，也将 numerical features 表示为相同的  $d$  维 low-dimensional space:

$$\mathbf{e}_m = \mathbf{v}_m x_m$$

其中：

- $\mathbf{v}_m$ : The embedding vector of  $m$ -th feature field (numerical field)
- $x_m$ : A scalar value
- $\mathbf{e}_m \in \mathbb{R}^d$ : The embedding vector of  $m$ -th feature field

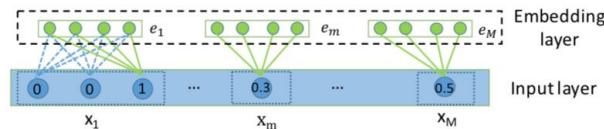


图 183: The input and embedding layer, both categorical and numerical fields are represented by low-dimensional dense vectors.

### 3. Interacting Layer

关键问题：如何建模高阶组合特征 (high-order combinatorial features) ?

- 传统方法：Domain experts knowledge  $\Rightarrow$  Createing meaningful combinations
- 本文方法：Multi-head self-attention mechanism  $\Rightarrow$  Modelling the correlations between different feature fields

Multi-head key-value self-attention mechanism 具体流程如下（以如何确定包含  $m$  特征的组合特征为例）：

(a) The correlation between feature  $m$  and feature  $k$  under a specific attention head  $h$ :

$$\alpha_{m,k}^{(h)} = \frac{\exp(\psi^{(h)}(\mathbf{e}_m, \mathbf{e}_k))}{\sum_{l=1}^M \exp(\psi^{(h)}(\mathbf{e}_m, \mathbf{e}_l))}$$

$$\psi^{(h)}(\mathbf{e}_m, \mathbf{e}_k) = \langle \mathbf{W}_{\text{Query}}^{(h)} \mathbf{e}_m, \mathbf{W}_{\text{Key}}^{(h)} \mathbf{e}_k \rangle \quad (\text{inner product})$$

注意：

- $\psi^{(h)}(\cdot, \cdot)$ : Attention function，定义特征 (不是域 field)  $m$  与  $k$  之间的相似度
- $\mathbf{W}_{\text{Query}}^{(h)}, \mathbf{W}_{\text{Key}}^{(h)} \in \mathbb{R}^{d' \times d}$ : Transformation matrices，将原始嵌入空间  $\mathbb{R}^d$  映射到新空间  $\mathbb{R}^{d'}$

(b) Update the representation of feature  $m$  in subspace  $h$  via combining all relevant features guided by coefficients  $\alpha_{m,k}^{(h)}$ :

$$\tilde{\mathbf{e}}_m^{(h)} = \sum_{k=1}^M \alpha_{m,k}^{(h)} (\mathbf{W}_{\text{Value}}^{(h)} \mathbf{e}_k)$$

其中：

- $\mathbf{W}_{Value}^{(h)} \in \mathbb{R}^{d' \times d}$
- $\tilde{\mathbf{e}}_m^{(h)} \in \mathbb{R}^{d'} : A combination of feature m and its relevant features (under head h),$  因此也代表了一个新的组合特征 (a new combinatorial feature)

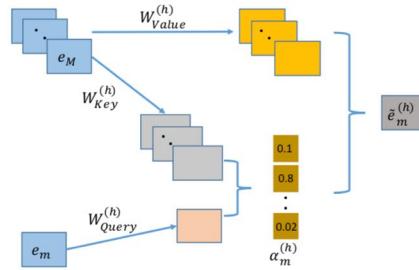


图 184: The architecture of interacting layer

- (c) Using multiple heads to create different subspaces and learn distinct feature interactions separately, collect combinatorial features learned in all subspaces:

$$\tilde{\mathbf{e}}_m = \tilde{\mathbf{e}}_m^{(1)} \oplus \tilde{\mathbf{e}}_m^{(2)} \oplus \dots \oplus \tilde{\mathbf{e}}_m^{(H)}$$

其中：

- $\oplus$ : Concatenation operator
- $H$ : The number of total heads
- $\tilde{\mathbf{e}}_m \in \mathbb{R}^{d'H}$

- (d) To preserve previously learned combinatorial features, including raw individual features, we add standard residual connections in our network:

$$\mathbf{e}_m^{\text{Res}} = \text{Relu}(\tilde{\mathbf{e}}_m + \mathbf{W}_{\text{Res}} \mathbf{e}_m)$$

其中：

- $\mathbf{W}_{\text{Res}} \in \mathbb{R}^{d'H \times d}$ : A projection matrix for dimension mismatching
- 这步操作实际上是一种 trade-off，即对于  $m$ -th 特征，同时考虑其原始 embedding :  $\mathbf{e}_m$ ，以及组合特征 :  $\tilde{\mathbf{e}}_m$
- $\mathbf{e}_m^{\text{Res}} \in \mathbb{R}^{d'H}$
- 未来可展开工作：
  - $\mathbf{W}_{\text{Res}} \mathbf{e}_m$  与  $\tilde{\mathbf{e}}_m$  两项的加权，譬如： $\rho \tilde{\mathbf{e}}_m + (1 - \rho) \mathbf{W}_{\text{Res}} \mathbf{e}_m$
  - 使用 average pooling, concatenation operator 等操作替换 sum pooling

- (e) We can model arbitrary-order combinatorial features by stacking multiple such layers

#### (f) Time Complexity Analysis

- $\because$  Calculating attention weights for one head:  $O(Mdd' + M^2d')$
- $\therefore O(MHd'(M + d))$ , It is therefore efficient because  $H$ ,  $d$  and  $d'$  are usually small.

#### 4. Output Layer

$$\hat{y} = \sigma(\mathbf{W}^T(e_1^{\text{Res}} \oplus e_2^{\text{Res}} \oplus \dots \oplus e_M^{\text{Res}}) + b)$$

其中：

- $\mathbf{W} \in \mathbb{R}^{d'HM}$
- $b \in \mathbb{R}$
- $\sigma$  is the sigmoid function

#### Loss function

$$\min_{\theta} -\frac{1}{N} \sum_{i=1}^N (y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})) \quad (\text{CTR Problem})$$

其中待学习参数集合  $\theta$  包括：

- $\mathbf{V}_j, j \in \{1, \dots, M\}$
- $\mathbf{W}_{\text{Query}}^{(h)}, \mathbf{W}_{\text{Key}}^{(h)}, \mathbf{W}_{\text{Value}}^{(h)}, h \in \{1, \dots, H\}$
- $\mathbf{W}_{\text{Res}}$
- $\mathbf{W}, b$

## 54.11 Convolutional Neural Networks based Click-Through Rate Prediction with Multiple Feature Sequences (未完成)

### 54.11.1 Introduction

本节来自论文：[Convolutional Neural Networks based Click-Through Rate Prediction with Multiple Feature Sequences\[?\]](#)

## 54.12 DeepGBM: A Deep Learning Framework Distilled by GBDT for Online Prediction Tasks (未完成)

### 54.12.1 Introduction

本节来自论文：[DeepGBM: A Deep Learning Framework Distilled by GBDT for Online Prediction Tasks\[?\]](#)

## 54.13 Sequence-Aware Factorization Machines for Temporal Predictive Analytics (SeqFM)

### 54.13.1 Introduction

论文出处：[Sequence-Aware Factorization Machines for Temporal Predictive Analytics\[?\]](#)

## 55 Deep Reinforcement Learning Approaches

### 55.1 DRN: A Deep Reinforcement Learning Framework for News Recommendation (未完成)

#### 55.1.1 Introduction

论文出处：[DRN: A Deep Reinforcement Learning Framework for News Recommendation\[?\]](#)

## 55.2 Deep Reinforcement Learning for List-wise Recommendations (未完成)

### 55.2.1 Introduction

本节选自论文Deep Reinforcement Learning for List-wise Recommendations[?]

## 55.3 Generative Adversarial User Model for Reinforcement Learning Based Recommendation System (未完成)

### 55.3.1 Introduction

论文出处：Generative Adversarial User Model for Reinforcement Learning Based Recommendation System[?]，来自ICML 2019

### 55.3.2 Model formulation

**Setting and RL formulation** 本模型基本的 setting 是：

- 首先，用户 (environment) 被系统 (agent) 提供展示一个包含  $k$  个 items 的页面
- 其次，用户以点击其中一个 item 或不点击任何 item 的形式向系统提供反馈信息
- 最后，系统推荐一个新的包含  $k$  个 items 的页面给用户

上述 setting 是对实际问题的一种最简单形式的抽象，因此便于用于之后的研究。而在文章所述的模型可以扩展到更为复杂的 setting 中。这里选择的建模方法是强化学习 (reinforcement learning)，选择原因是因为强化学习会考虑长期收益 (long-term reward)，因此可以帮助改进用户与在线平台长期互动 (user's long-term engagement with an online platform)。

因此推荐系统旨在发现一种策略  $\pi(s, \mathcal{I})$ ，用于根据用户状态  $s$ ，输出在所有 items  $\mathcal{I}$  中的选择概率分布，从而最大化推荐系统 (agent) 自身针对用户的长期期望收益 (long-term expected reward, e.g. user's interest of platform)

$$\pi^* = \arg \max_{\pi(s^{(t)}, \mathcal{I}^{(t)})} \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^{(t)} r(s^{(t)}, a^{(t)}) \right]$$

其中：

- $s^{(0)} \sim p^{(0)}$
- $s^{(t+1)} \sim P(\cdot | s^{(t)}, \mathcal{A}^{(t)})$
- $\mathcal{A}^{(t)} \sim \pi(s^{(t)}, \mathcal{I}^{(t)})$
- $a^{(t)} \in \mathcal{A}^{(t)}$

嵌入 Reinforcement Learning 的框架，每个部分解释如下：

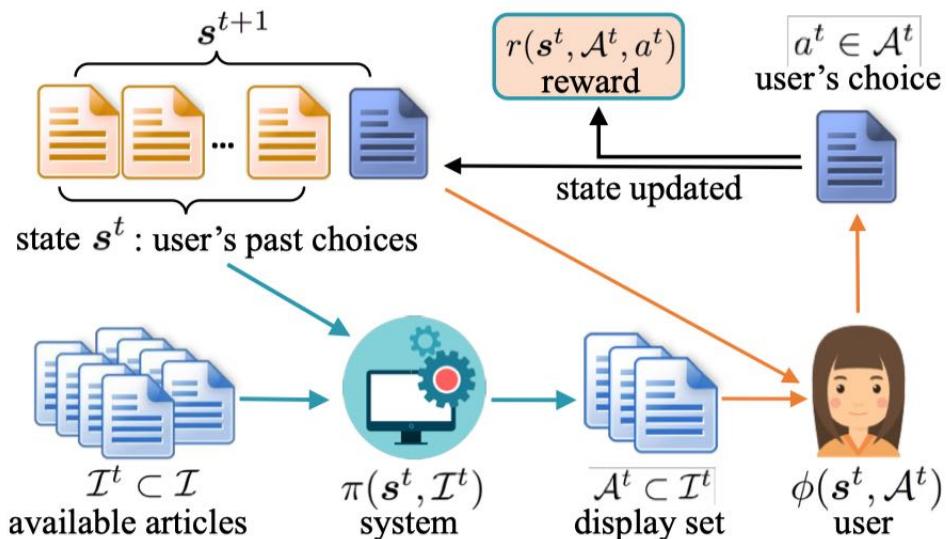


图 185: Illustration of the interaction between a user and the recommendation system. Green arrows represent the recommender information flow and orange represents user's information flow.

- Environment
- State
- Action
- State Transition
- Reward Function
- Policy

**Generative Adversarial User Model**

**Generative Adversarial Training**

## 56 Generative Model Approaches

### 56.1 Collaborative Variational Autoencoder for Recommender Systems（未完成）

#### 56.1.1 Introduction

本文选自Collaborative Variational Autoencoder for Recommender Systems[?]

**Part XII****Model Applications of System Security**

## 57 Generative Adversarial Networks in Face Recognition & Detection

### 57.1 Adversarial Attacks on Face Detectors using Neural Net based Constrained Optimization (未完成)

#### 57.1.1 Introduction

本节选自论文Adversarial Attacks on Face Detectors using Neural Net based Constrained Optimization[?]

## 57.2 ADVHAT: Real-World Adversarial Attrack on ARCFACE Face ID System (未完成)

### 57.2.1 Introduction

本节选自论文AdvHat: Real-world adversarial attack on ArcFace Face ID system[?]

### 57.2.2 Reference

- reddit: [R] AdvHat: Real-world adversarial attack on ArcFace Face ID system

## 58 Generative Adversarial Networks in Fingerprint Recognition & Detection (未完成)

## Part XIII

# Model Applications of Transportation

## 59 Order Dispatch

### 59.1 Large-Scale Order Dispatch in On-Demand Ride-Hailing Platforms: A Learning and Planning Approach

#### 59.1.1 Introduction

选自论文Large-Scale Order Dispatch in On-Demand Ride-Hailing Platforms: A Learning and Planning Approach[?]

#### Background

- traditional taxis "driver-select-order mode" to a centralized "platform-assign-order-to-driver mode" leads to a significant improvement on the platform's efficiency, resulting in a more than 10% improvement on the order completion rate.
- Order dispatch, which aims to find the best matching between drivers and orders, is obviously crucial for on-demand ride-hailing services (按需打车服务) .

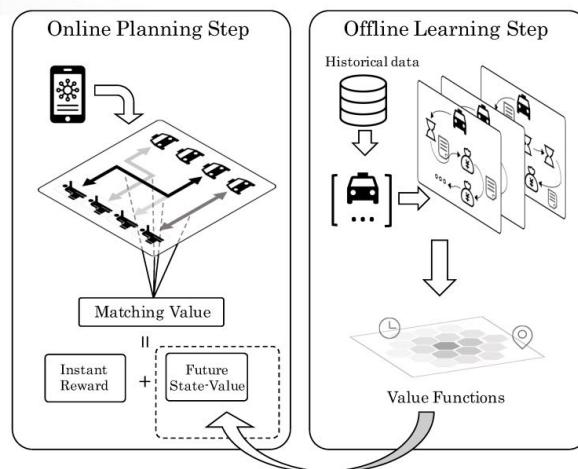


图 186: Architecture of order dispatch in on-demand ride-hailing services.

- Equipped with sensing and communication tools, each taxi periodically uploads its geographical coordinates and occupancy status to the platform.
- when a passenger generates a request "on-demand", he or she will place an order on the platform right away.

#### 59.1.2 Model formulation

##### Learning

##### Planning

## Part XIV

# Numerical Optimization and Operational Research Theory and Algorithms

# 60 Optimization Overview

## 60.1 Overview

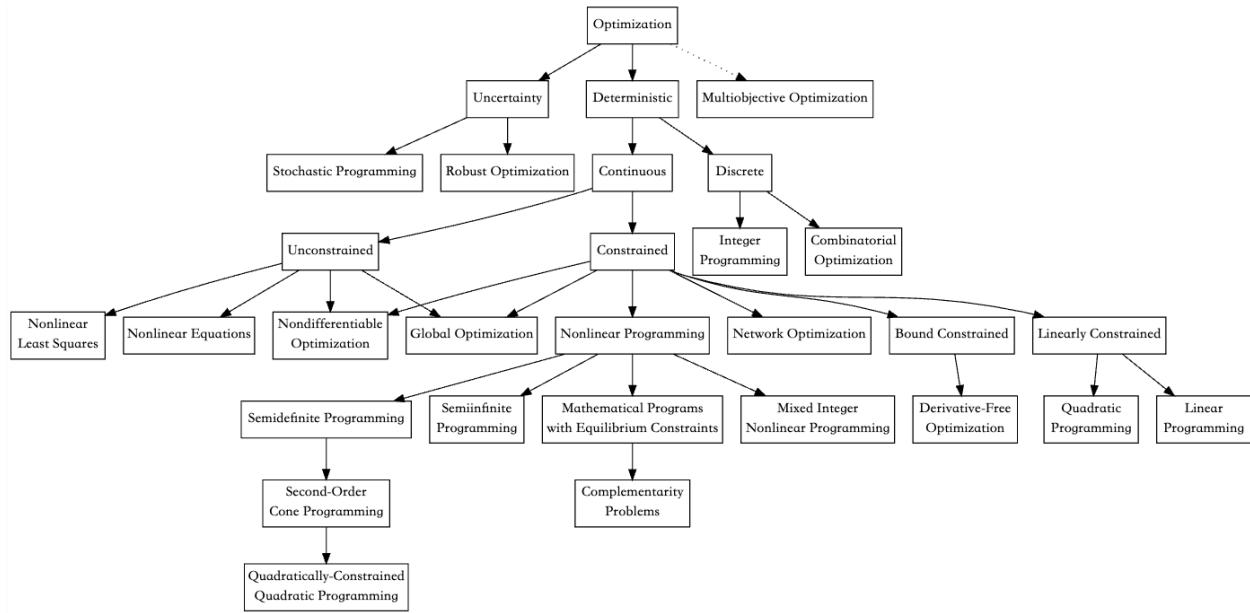


图 187: Categorize your optimization problem. Cite from STA 663: Computational Statistics and Statistical Computing

## 60.2 Reference

- STA 663: Computational Statistics and Statistical Computing (2018)

# 61 Unconstrained Optimization

## 61.1 Line Search Methods

### 61.1.1 General description

- Each iteration of a line search method contains:
  1. compute a **search direction**  $p_k$
  2. decide how far to move along that direction  $\alpha_k$

The iteration is given by:

$$x_{k+1} = x_k + \alpha_k p_k$$

where the positive scalar  $\alpha_k$  is called the **step length**.

- The success of a line search method depends on effective choice of both the direction  $p_k$  and the step length  $\alpha_k$ . In this chapter, we discuss:
  - How to choose  $\alpha_k$  and  $p_k$  to promote convergence from remote starting points;
  - Study the convergence results of several popular Line search algorithms.

### 61.1.2 Choice of search direction

**Theorem 10 (Taylor's Theorem).** Suppose that  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is a continuously differentiable and that  $p \in \mathbb{R}^n$ . Then we have that:

$$f(x + p) = f(x) + \nabla f(x + tp)^T p,$$

for some  $t \in (0, 1)$ . Moreover, if  $f$  is twice continuously differentiable, we have that:

$$\nabla f(x + p) = \nabla f(x) + \int_0^1 \nabla^2 f(x + tp)p dt$$

and then:

$$f(x + p) = f(x) + p^T \nabla f(x) + \frac{1}{2} p^T \nabla^2 f(x + tp) p$$

for some  $t \in (0, 1)$ .

**Definition 61.1 (positive definite matrix).** In linear algebra, a symmetric  $n \times n$  real matrix  $M$  is said to be positive definite if the scalar  $x^T M x$  is strictly positive for every non-zero column vector  $x$  of  $n$  real numbers. Here  $x^T$  denotes the transpose of  $x$ . Formally:

$$M \text{ is positive definite} \iff x^T M x > 0 \text{ for all } x \in \mathbb{R}^n \setminus \mathbf{0}$$

**Definition 61.2 (positive semi-definite matrix).** In linear algebra, a symmetric  $n \times n$  real matrix  $M$  is said to be positive semidefinite or non-negative definite if  $x^T M x \geq 0$  for all  $x \in \mathbb{R}^n$ , Formally:

$$M \text{ is positive semi-definite} \iff x^T M x \geq 0 \text{ for all } x \in \mathbb{R}^n$$

### Steepest Descent Direction for Line Search Methods

- Consider the Taylor's theorem, which tells us that for any search direction  $p$  and step-length parameter  $\alpha$ , we have:

$$f(x_k + \alpha p) = f(x_k) + \alpha p^T \nabla f(x_k) + \frac{\alpha^2}{2} p^T \nabla^2 f(x_k + tp)p,$$

for some  $t \in (0, \alpha)$ .

- Consider only the first order information, we can get the residual of objective to be minimized between source point and ideal target point as follow:

$$f(x_k) - f(x_k + \alpha p) = -\alpha p^T \nabla f(x_k)$$

therefore we only need to maximize this residual:

$$\begin{aligned} & \max_{\alpha, p} f(x_k) - f(x_k + \alpha p) \\ & \Rightarrow \max_{\alpha, p} -\alpha p^T \nabla f(x_k) \\ & \Rightarrow \min_{\alpha, p} \alpha p^T \nabla f(x_k) \end{aligned}$$

- The rate of change in  $f$  along the direction  $p$  at  $x_k$  is simply the coefficient of  $\alpha$ . Hence, we use the unite direction  $p$  for only considering direction selection. The unite direction  $p$  of most rapid decrease is the solution to the problem:

$$\min_p p^T \nabla f(x_k), \text{ subject to } \|p\| = 1.$$

- Since  $p^T \nabla f(x_k) = \|p\| \|f(x_k)\| \cos \theta$ , where  $\theta$  is the angle between  $p$  and  $\nabla f(x_k)$ , it is easy to see that the minimizer is attained when  $\cos \theta = -1$  and

$$p = -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}$$

Therefore,  $-\nabla f(x_k)$  is the one along which  $f$  decreases most rapidly.

- Advantages:

– It only requires calculation of the gradient  $\nabla f(x_k)$  but not of second derivatives.

- Disadvantages:

– It can be excruciatingly slow to convergence on difficult problems.

**In general**(the steepest descent direction is a special case), **any descent direction  $p$ , one that satisfies:**

$$p^T \nabla f(x_k) < 0, \exists p \in \mathbb{R}^n$$

is guaranteed to produce a decrease in  $f$ , provided that the step length is sufficiently small. Actually, by using Taylor's theorem we have that:

$$f(x_k + \epsilon p_k) = f(x_k) + \epsilon p_k^T \nabla f(x_k) + O(\epsilon^2)$$

When  $p_k$  is a descent direction, it follows that  $f(x_k + \epsilon p) < f(x_k)$  for **sufficiently small positive values of  $\epsilon$** .

**Newton Direction for Line Search Methods** Consider the second-order Taylor series approximation to  $f(x_k + p)$ , which is:

$$f(x_k + p) \approx f(x_k) + p^T \nabla f(x_k) + \frac{1}{2} p^T \nabla^2 f(x_k) p \equiv m_{x_k}(p)$$

Assuming here  $\nabla^2 f(x_k)$  is **positive definite**, therefore the Newton direction is obtained by finding the vector  $p$  that minimizes  $m_{x_k}(p)$ . In detail, by simply setting the derivatives of  $m_{x_k}(p)$  to zero:

$$\begin{aligned}\nabla_p m_{x_k}(p) &= 0 \\ \Rightarrow \nabla f(x_k) + \nabla^2 f(x_k) p &= 0\end{aligned}$$

we obtain the following explicit formula for the Newton direction:

$$p = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

#### Q.A. Why the Newton direction is a descent direction?

- The Newton direction can be used in a line search method when  $\nabla^2 f(x_k)$  is positive definite, for in this case we have:

$$\nabla^T f(x_k) p_k = -\nabla^T f(x_k) (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

#### Quasi-Newton Direction for Line Search Methods

- Quasi-Newton search directions provides an attractive alternative to Newton's method in which they do not require computation of the Hessian and yet still attain a superlinear rate of convergence.
- Use  $B_k$  to approximate the true Hessian  $\nabla^2 f(x_k)$ . Define quasi-Newton direction:

$$p_k = -B_k^{-1} \nabla f(x_k)$$

- Update  $B_k$  after each step to take account of the additional knowledge gained during the step. The updates make use of the fact that changes in the gradient  $g$  provide information about the second derivative of  $f$  along the search direction.

### 61.1.3 Choice of step length

In computing the step length  $\alpha_k$ , we face a tradeoff.

- We would like to choose a good step length  $\alpha_k$  to give a substantial reduction of  $f$ ;
- but at the same time we do not want to spend too much time making the choice.

#### Exact line search

- The ideal choice would be the global minimizer of the univariate function  $\phi(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$  defined by:

$$\min_{\alpha} \phi(\alpha) = f(x_k + \alpha p_k), \alpha > 0.$$

where  $p_k$  is solved search direction(constant).

- But in general, it is too expensive to identify this value. To find even a local minimizer of  $\phi$  to moderate precision generally requires too many evaluations of the objective function  $f$  and possibly the gradient  $\nabla f(x_k)$ .

#### Inexact line search

**61.1.4 Global convergence of line search methods**

## 61.2 Trust Region Methods

## 62 Constrained Optimization

### 62.1 Interior-Point Methods for Nonlinear Programming

## 62.2 Branch and Bound Methods for Integer Programming

### 62.2.1 Reference

- Integer Programming: The Branch and Bound Method

## 63 Robust Optimization

### 63.1 A Practical Guide to Robust Optimization

本节内容主要参考自论文A Practical Guide to Robust Optimization[?]

## 63.2 Distributionally Robust Optimization under Moment Uncertainty with Application to Data-Driven Problems

论文出处：[Distributionally Robust Optimization under Moment Uncertainty with Application to Data-Driven Problems\[?\]](#)

### 63.3 Reference

- Theory and applications of Robust Optimization[?]
- When Machine Learning Meets Robust Optimization –DDARO Models, Algorithms and Applications for Industry 4.0

## 64 Stochastic Optimization

## 65 Operational Research Models

### 65.1 Hub Location

论文出处 : Hub location problems: A review of models, classification, solution techniques, and applications[?]

## Part XV

# Mathematical Basis in Machine Learning

## 66 Mathematical Basis

### 66.1 Probability & Information Theory

#### 66.1.1 Beta distribution

**Probability density function** Beta 分布可视为概率的概率分布，其概率密度函数（pdf）如下：

$$f(x; \alpha, \beta) = \underbrace{\frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)}}_{\text{const}} x^{\alpha-1} (1-x)^{\beta-1} \quad (x \in [0, 1])$$

其中  $\Gamma$  为 gamma 函数：

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$$

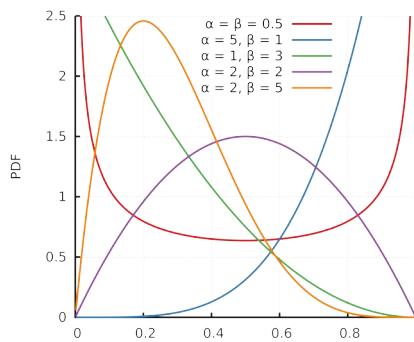


图 188: The probability density function of beta-distribution

#### Properties

- Mean:  $\frac{\alpha}{\alpha+\beta}$
- Stddev:  $\frac{\sqrt{\alpha\beta}}{(\alpha+\beta)^2(\alpha+\beta+1)}$

#### Reference

- Beta distribution - Wikipedia
- 如何通俗理解 beta 分布 ? - 知乎

### 66.1.2 Wasserstein distance

#### Reference

- Wasserstein Distributionally Robust Optimization: Theory and Applications in Machine Learning
- Wasserstein GAN and the Kantorovich-Rubinstein Duality

## 66.2 Convex Optimization

### 66.2.1 Karush–Kuhn–Tucker Conditions (KKT)

考虑如下的一个约束优化问题：

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & g_i(\boldsymbol{x}) \leq 0, \quad \forall i \in \{1, \dots, m\} \\ & h_j(\boldsymbol{x}) = 0, \quad \forall j \in \{1, \dots, l\} \end{aligned}$$

该问题的拉格朗日函数为：

$$J(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\boldsymbol{x}) + \sum_{i=1}^m \alpha_i g_i(\boldsymbol{x}) + \sum_{j=1}^l \beta_j h_j(\boldsymbol{x})$$

因此问题就转换为了：

$$\boldsymbol{x}^* = \arg \min_{\boldsymbol{x}} J(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

KKT 条件是非线性规划最优解的**必要条件**，特别地，当问题是凸问题时候，KKT 条件是最优解的充要条件。

KKT 条件如下：

$$\left\{ \begin{array}{l} \nabla_{\boldsymbol{x}} = 0 \\ g_i(\boldsymbol{x}) \leq 0, \quad \forall i \in \{1, \dots, m\} \\ h_j(\boldsymbol{x}) = 0, \quad \forall j \in \{1, \dots, l\} \\ \alpha_i \geq 0, \quad \forall i \in \{1, \dots, m\} \\ \alpha_i g_i(\boldsymbol{x}) = 0 \quad \forall i \in \{1, \dots, m\}. \end{array} \right.$$

## 66.3 Linear Algebra

### 66.3.1 Vector Gradient

#### Vector-wise Vector Gradient

$$\begin{aligned}\frac{\partial \mathbf{x}}{\partial \mathbf{x}} &= \mathbf{I} \\ \frac{\partial \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} &= \mathbf{A}^T \\ \frac{\partial \mathbf{x}^T \mathbf{A}}{\partial \mathbf{x}} &= \mathbf{A}\end{aligned}$$

**Bit-wise Vector Gradient** 假设一个函数  $f(x)$  的输入是一个标量  $x$ 。对于一组  $K$  个标量  $x_1, \dots, x_K$ ，我们可以通过函数  $f(x)$  对每一个元素  $x_k$  操作得到另一组标量  $z_1, \dots, z_K$ ，即：

$$z_k = f(x_k), \quad \forall k \in \{1, \dots, K\}$$

为了简便起见，我们定义  $\mathbf{x} = (x_1, \dots, x_K)^T$ ,  $\mathbf{z} = (z_1, \dots, z_K)^T$ ，那么：

$$\mathbf{z} = f(\mathbf{x})$$

其中  $f(\mathbf{x})$  是按位运算的，即： $[f(\mathbf{x})]_i = f(x_i)$ 。

- 当  $x$  为标量时： $f(x)$  的导数记作  $f'(x)$ ；
- 当  $\mathbf{x}$  为向量时： $f(\mathbf{x})$  关于  $\mathbf{x}$  的导数为一个对角矩阵：

$$\begin{aligned}\frac{df(\mathbf{x})}{d\mathbf{x}} &= \left[ \frac{df(x_j)}{dx_j} \right]_{K \times K} \\ &= \begin{bmatrix} f'(x_1) & 0 & \cdots & 0 \\ 0 & f'(x_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'(x_K) \end{bmatrix} \\ &= \text{diag}(f'(\mathbf{x}))\end{aligned}$$

注意此求梯度式子一般用在同一层激活函数作用下的输入和输出之间求梯度，因为激活函数的作用是每一个维度相互独立的，即 bit-wise 的。

#### Note

- 参数（链接线）的求导是 vector-wise 的
- 激活函数的求导是 bit-wise 的

#### Reference

- The Matrix Calculus You Need For Deep Learning

### 66.3.2 Tensor Operations

**Matrix-by-tensor multiply operation** 譬如：在 LSTM 时序分类任务中，每一时刻的隐层批输出  $\mathbf{h}_t \in \mathbb{R}^{m \times D}$  都需要对应一个类别概率分布输出  $\mathbf{z}_t \in \mathbb{R}^{m \times K}$ ，则：

- 隐层批时序输出： $\mathbf{h} \in \mathbb{R}^{m \times T \times D}$
- 隐层  $\Rightarrow$  输出层参数： $\mathbf{W} \in \mathbb{R}^{D \times K}$

做法如下：

1. 数据 Tensor  $\Rightarrow$  数据 Matrix： $\mathbf{h} \in \mathbb{R}^{m \times T \times D} \Rightarrow \mathbf{h}' \in \mathbb{R}^{(m \times T) \times D}$
2. 数据 Matrix 与参数 Matrix 的矩阵乘法： $\tilde{\mathbf{y}}' = \mathbf{h}' \mathbf{W} \in \mathbb{R}^{(m \times T) \times K}$
3. 输出 Matrix  $\Rightarrow$  输出 Tensor： $\hat{\mathbf{y}}' \in \mathbb{R}^{(m \times T) \times K} \Rightarrow \hat{\mathbf{y}} \in \mathbb{R}^{m \times T \times K}$

即：**计算前合并，计算后分解与矩阵计算不相关的维度。**

## Part XVI

# Software of Machine Learning

## 67 Deep Learning Framework

### 67.1 TensorFlow

#### 67.1.1 Multi-GPUs

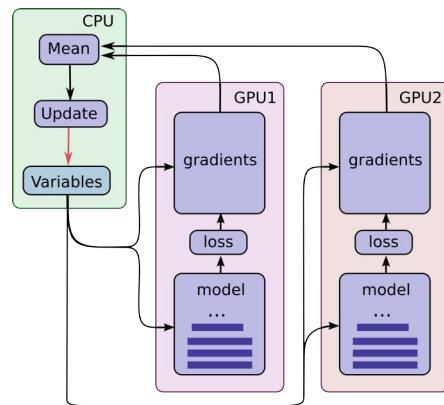


图 189: TensorFlow[?] Multi-GPU

#### Topology

#### Materials

- CIFAR-10 Multi-GPU tutorial

### 67.1.2 Multi-Workers

#### Materials

- Distributed TensorFlow

### 67.1.3 TensorFlow Serving

#### Using TensorFlow Serving with Docker

1. Installing Docker
2. Pulling a serving image: docker pull tensorflow/serving[?]
3. Start TensorFlow Serving container and open the rest API port using bash shell:

```
#!/bin/bash
docker run -p 8501:8501 \
--mount type=bind,source=<MODEL_DIR>,target=/models/<MODEL_NAME> \
-e MODEL_NAME=<MODEL_NAME> -t tensorflow/serving
```

4. Query the model using the predict API:

```
Example-1: one sample, one input field
curl -d '{"instances": [[1.1,1.2,0.8,1.3]]}' -X POST
http://localhost:8501/v1/models/<MODEL_NAME>:predict
```

```
Example-2: multi samples, multi input fields
curl -d "{
 \"instances\": [
 {
 \"inds\": [0,4,13,64,68,77,83,87,90,91,95,98,109],
 \"vals\": [0.441519,1,1,1,1,1,1,1,1,1,1,1,1]
 },
 {
 \"inds\": [0,4,13,64,68,77,83,87,90,91,95,100,107],
 \"vals\": [0.221519,1,1,1,1,1,1,1,1,1,1,1,1]
 }
]
}" -X POST http://localhost:8501/v1/models/DeepFM:predict
```

5. Quit docker container:

```
docker ps
docker stop <CONTAINER ID>
```

#### 67.1.4 TensorFlow Estimator

**Problem** 深度学习模型结构层出不穷，如何验证其有效性，快速落地于工业界存在以下一些问题：

- 学术界的开源实现距离工业应用水平还有较大的差距
- 模型实现大量调用底层、低版本 API，兼容性和高效性存在问题
- 单机实现无法应用于工业场景
- 迁移成本较高，边际成本高，每种算法都需要从头开始搭建

**Solution** 针对上述存在的问题，先提出一套基于 TensorFlow Estimator[?] 的解决方案：

- 数据读取 `input_fn` 采用 TensorFlow Dataset API，支持高效队列、并行的读取方式
- 模型编写通过 TensorFlow Estimator API 封装算法  $f(x)$ ，实验新算法边际成本比较低，只需要改写 `model_fn` 中  $f(x)$  部分
- 支持分布式以及单机多线程训练
- 支持 `export model`，然后用 TensorFlow Serving 提供线上预测服务

**Example** 具体解决方案，查看 CNN 一章 LeNet5 的实现

### 67.1.5 Note

#### tf.identity

- $y = \text{tf.identity}(x)$ ：张量  $y$  和  $x$  之间有边的关系，等价于  $y = x + 0.0$
- $y = x$ ： $y$  是一个复制了  $x$  变量的变量，并未和  $x$  计算图上的节点相联系，不接受流程控制函数的调遣

**tf.assign** 一般为针对自增的  $\text{trainable} = \text{False}$  的变量进行的操作，同时配合：

---

```
import tensorflow as tf

with tf.control_dependencies(control_inputs=[assign_op ...]):
 ...
```

---

进行使用。

## 67.2 PyTorch

### 67.2.1 Multi-GPUs

### 67.2.2 Multi-Workers

### 67.3 Keras

## 67.4 Horovod

Horovod[?] 是 Uber 开源的分布式学习框架，可在 TensorFlow 和 PyTorch 之上进行包装和优化。

### 67.4.1 Reference

- TensorFlow Distributed Training: Introduction and Tutorials

## 67.5 TensorRT

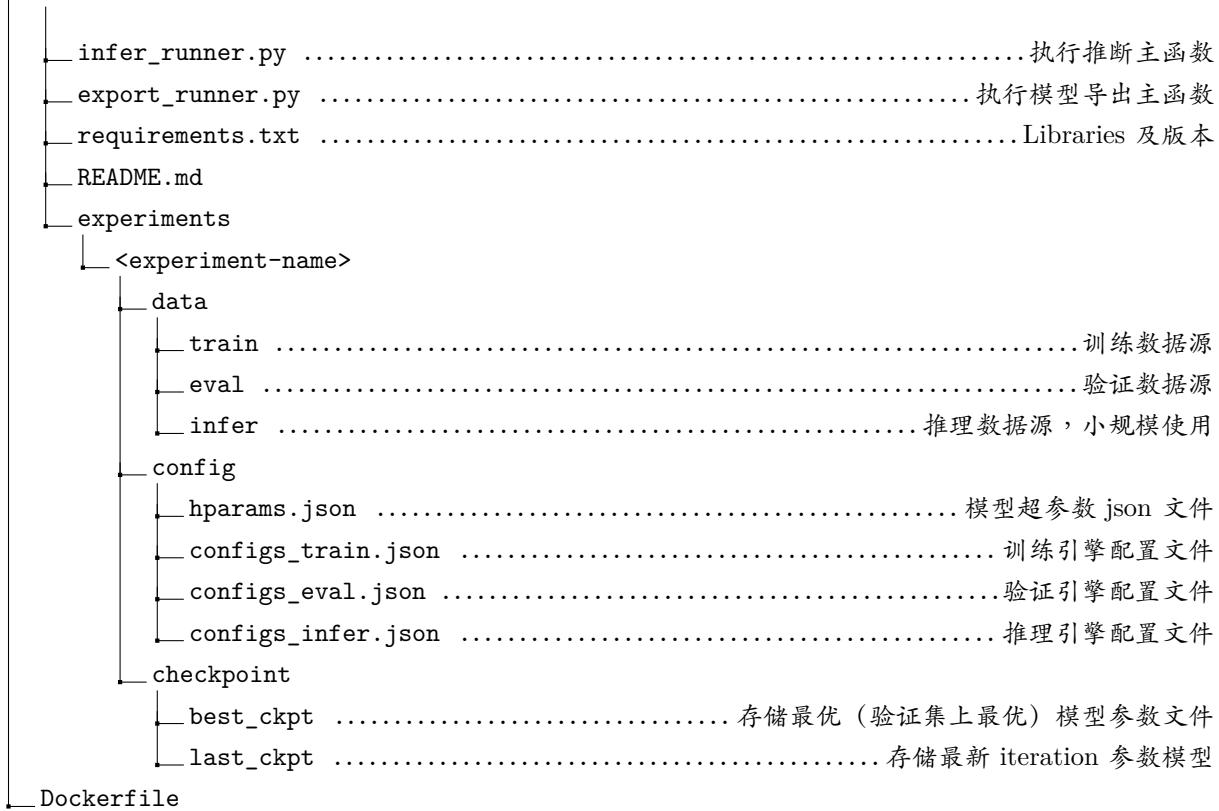
TensorRT用于加速模型在测试阶段的 inference 效率，更好地满足大规模应用部署的需求。

## 68 TensorFlow Project Template

### 68.1 Overall Project Architecture

TensorFlow 机器学习项目（以 AlexNet 图像分类项目为例）的模型文件目录（文件架构如下）：





### 68.1.1 Problem

1. 如何设置采样策略 (e.g. 训练样本的 importance sampling) ?
2. 超参数如何调节 ?
3. 强化学习和生成对抗网络怎么办 ?

### 68.1.2 Reference

- [cs230-stanford/cs230-code-examples: Hand Signs Recognition with Tensorflow](#)

## 68.2 Model

### **68.3 Engine**

## 68.4 Utils

## 69 Operational Research Framework

### 69.1 Mosek

#### 69.1.1 Linear programming

```

-*- coding: utf-8 -*-
"""
Created on 2018/10/26 14:18
author: Tong Jia
email: cecilio.jia@gmail.com
software: PyCharm
Description:
 An implement of linear programming using Mosek framework.
bound key of variables and constraints:
 1. mosek.boundkey.fx [a, a]
 2. mosek.boundkey.lo [a, +inf)
 3. mosek.boundkey.up (-inf, b]
 4. mosek.boundkey.ra [a, b]
Optimizer type:
 mosek.soltype.bas (Linear programming)
 mosek.soltype.itr (Linear programming, Quadratic programming)
 mosek.soltype.itg (Mixed integer programming)
"""

import mosek
import sys
from six.moves import xrange

Since the value of infinity is ignored, we define it solely
for symbolic purposes
inf = 0.0

Define a stream printer to grab output from MOSEK
def streamprinter(text):
 sys.stdout.write(text)
 sys.stdout.flush()

def model():
 # Make mosek environment
 with mosek.Env() as env:
 # Create a task object
 with env.Task(0, 0) as task:
 # Attach a log stream printer to the task
 task.set_Stream(mosek.streamtype.log, streamprinter)

 # Bound keys for constraints
 bkc =[mosek.boundkey.fx, mosek.boundkey.lo, mosek.boundkey.up]

 # Bound values for constraints
 blc =[30.0, 15.0, -inf]
 buc =[30.0, +inf, 25.0]

```

```

Bound keys for variables
bkx = [
 mosek.boundkey.lo,
 mosek.boundkey.ra,
 mosek.boundkey.lo,
 mosek.boundkey.lo
]

Bound values for variables
blx =[0.0, 0.0, 0.0, 0.0]
bux =[+inf, 10.0, +inf, +inf]

Objective coefficients
c =[3.0, 1.0, 5.0, 1.0]

Below is the sparse representation of the A
matrix stored by column.
asub =[[0, 1],
 [0, 1, 2],
 [0, 1],
 [1, 2]]
aval =[[3.0, 2.0],
 [1.0, 1.0, 2.0],
 [2.0, 3.0],
 [1.0, 3.0]]

numvar =len(bkx)
numcon =len(bkc)

Append 'numcon' empty constraints.
The constraints will initially have no bounds.
task.appendcons(numcon)

Append 'numvar' variables.
The variables will initially be fixed at zero (x=0).
task.appendvars(numvar)

for j in xrange(numvar):
 # Set the linear term c_j in the objective.
 task.putcj(j, c[j])

 # Set the bounds on variable j
 # blx[j] <= x_j <= bux[j]
 task.putvarbound(j, bkx[j], blx[j], bux[j])

 # Input column j of A
 task.putacol(j, # Variable (column) index.
 asub[j], # Row index of non-zeros in column j.
 aval[j]) # Non-zero Values of column j.

 # Set the bounds on constraints.
 # blc[i] <= constraint_i <= buc[i]

```

```
for i in xrange(numcon):
 task.putconbound(i, bkc[i], blc[i], buc[i])

Input the objective sense (minimize/maximize)
task.putobjsense(mosek objsense maximize)

Solve the problem
task.optimize()

Print a summary containing information
about the solution for debugging purposes
task.solutionsummary(mosek.streamtype.msg)

Get status information about the solution
solsta = task.getsolsta(mosek.soltype.bas)

if (solsta ==mosek.solsta.optimal or
 solsta ==mosek.solsta.near_optimal):
 xx = [0.] *numvar
 task.getxx(mosek.soltype.bas, # Request the basic solution.
 xx)
 print("Optimal solution: ")
 for i in range(numvar):
 print("x["+ +str(i) +"]=" +str(xx[i]))
elif (solsta ==mosek.solsta.dual_infeas_cer or
 solsta ==mosek.solsta.prim_infeas_cer or
 solsta ==mosek.solsta.near_dual_infeas_cer or
 solsta ==mosek.solsta.near_prim_infeas_cer):
 print("Primal or dual infeasibility certificate found.\n")
elif solsta ==mosek.solsta.unknown:
 print("Unknown solution status")
else:
 print("Other solution status")

if __name__ =='__main__':
 model()
```

## 69.2 Gurobi

### 69.2.1 Mixed integer linear programming

```
-*- coding: utf-8 -*-
"""
Created on 2018/3/8 12:12
author: Tong Jia
email: cecilio.jia@gmail.com
software: PyCharm
Description:
 A basic implement of mip using gurobi callback solver.
 This example formulates and solves the following simple MIP model:
 maximize x + y + 2z
 subject to x + 2y + 3z <=4
 x + y >=1
 x, y, z belong to {0, 1}
"""

import gurobipy
from six.moves import xrange

def mipSolver(numConstraints,
 boundKeysConstraintsVec, boundValuesConstraintsVec,
 numVariables,
 nameVariablesVec, boundKeysVariablesVec, lowerboundValuesVariablesVec,
 upperboundValuesVariablesVec,
 objLinearCoefficientsVec,
 constraintsCoefficientsMatrix):
 """
 Parameters:

 :param numConstraints: int
 :param boundKeysConstraintsVec: list
 :param boundValuesConstraintsVec: list
 :param numVariables: int
 :param nameVariablesVec: list
 :param boundKeysVariablesVec: list
 :param lowerboundValuesVariablesVec: list
 :param upperboundValuesVariablesVec: list
 :param objLinearCoefficientsVec: list
 :param constraintsCoefficientsMatrix: list

 Returns:

 :return: solutionsDict: dict
 :return: objValue: float
 """
 model =gurobipy.Model("mixed integer programming")

 # Add variables to model
 variables =[]
 for j in xrange(numVariables):
 """
 Create a binary variable
 """
 variables.append(model.addVar(vtype=gurobipy.GRB.BINARY))
 model.update()

 # Set objective
 model.setObjective(objLinearCoefficientsVec, gurobipy.GRB.MAXIMIZE)

 # Add constraint
 for i in xrange(numConstraints):
 """
 Create a constraint
 """
 boundKey = boundKeysConstraintsVec[i]
 boundValue = boundValuesConstraintsVec[i]
 if boundKey == "L":
 model.addConstr(variables[boundValue] >= 1)
 elif boundKey == "U":
 model.addConstr(variables[boundValue] <= 1)
 else:
 model.addConstr(variables[boundValue] == 1)
 model.update()

 # Optimize
 model.optimize()
 print("Optimal value: %s" % model.ObjVal)
 print("Optimal solution: %s" % model.getAttr('X', variables))

 # Get solution
 solutionsDict = {}
 for v in variables:
 solutionsDict[v.VarName] = v.X
 return solutionsDict, model.ObjVal
```

```

variables.append(
 model.addVar(
 lb=lowerboundValuesVariablesVec[j],
 ub=upperboundValuesVariablesVec[j],
 obj=objLinearCoefficientsVec[j],
 vtype=boundKeysVariablesVec[j],
 name=nameVariablesVec[j]
)
)

Populate constraints matrix
for i in xrange(numConstraints):
 expr =gurobipy.LinExpr()
 for j in xrange(numVariables):
 if (constraintsCoefficientsMatrix[i][j] !=0):
 expr +=constraintsCoefficientsMatrix[i][j] *variables[j]
 model.addConstr(
 lhs=expr,
 sense=boundKeysConstraintsVec[i],
 rhs=boundValuesConstraintsVec[i],
)

Populate objective
Add linear terms
objective =gurobipy.LinExpr()
for j in xrange(numVariables):
 if (objLinearCoefficientsVec[j] !=0):
 objective +=objLinearCoefficientsVec[j] *variables[j]

parameter sense in function model.setObjective takes values from {gurobipy.GRB.MAXIMIZE,
gurobipy.GRB.MINIMIZE}
model.setObjective(objective, gurobipy.GRB.MAXIMIZE)

Solve the model
model.optimize()

Print solutions
if (model.status ==gurobipy.GRB.Status.OPTIMAL):
 solutionsDict ={}
 for variable_index in xrange(len(model.getVars())):
 variable =model.getVars()[variable_index]
 solutionsDict[nameVariablesVec[variable_index]] =round(variable.x)
 objValue =model.objVal
 return solutionsDict, objValue
elif (model.status ==gurobipy.GRB.Status.INF_OR_UNBD):
 print("Model is infeasible or unbounded")
 exit(0)
elif (model.status ==gurobipy.GRB.Status.INFEASIBLE):
 print("Model is infeasible")
 exit(0)
elif (model.status ==gurobipy.GRB.Status.UNBOUNDED):
 print("Model is unbounded")

```

```
 exit(0)
 else:
 print("Optimization ended with status %d" % model.status)
 exit(0)

if __name__ == '__main__':
 numConstraints = 2
 boundKeysConstraintsVec =[gurobipy.GRB.LESS_EQUAL, gurobipy.GRB.GREATER_EQUAL]
 boundValuesConstraintsVec =[4.0, 1.0]
 numVariables = 3
 nameVariablesVec =["x", "y", "z"]
 boundKeysVariablesVec =[gurobipy.GRB.BINARY, gurobipy.GRB.BINARY, gurobipy.GRB.BINARY]
 lowerboundValuesVariablesVec =[0.0, 0.0, 0.0]
 upperboundValuesVariablesVec =[1.0, 1.0, 1.0]
 objLinearCoefficientsVec =[1, 1, 2]
 constraintsCoefficientsMatrix =[

 [1, 2, 3],
 [1, 1, 0]
]

 try:
 solutionsDict, objValue =mipSolver(
 numConstraints=numConstraints,
 boundKeysConstraintsVec=boundKeysConstraintsVec,
 boundValuesConstraintsVec=boundValuesConstraintsVec,
 numVariables=numVariables,
 nameVariablesVec=nameVariablesVec,
 boundKeysVariablesVec=boundKeysVariablesVec,
 lowerboundValuesVariablesVec=lowerboundValuesVariablesVec,
 upperboundValuesVariablesVec=upperboundValuesVariablesVec,
 objLinearCoefficientsVec=objLinearCoefficientsVec,
 constraintsCoefficientsMatrix=constraintsCoefficientsMatrix
)
 except gurobipy.GurobiError as e:
 print('Error code ' +str(e.errno) +": " +str(e))
 except AttributeError:
 print('Encountered an attribute error')
```

### 69.2.2 Quadratic programming

```

-*- coding: utf-8 -*-
"""
Created on 2018/3/7 10:16
author: Tong Jia
email: cecilio.jia@gmail.com
software: PyCharm
Description:
 A basic implement of qp using gurobi callback solver.
This example formulates and solves the following simple QP model:
 minimize x + y + x^2 + x*y + y^2 + y*z + z^2
 subject to x + 2y + 3z >= 4
 x + y >= 1
"""

import gurobipy
from six.moves import xrange

def qpSolver(numConstraints, numVariables, nameVariablesVec,
 objLinearCoefficientsVec, objQuadraticCoefficientsMatrix,
 constraintsCoefficientsMatrix,
 boundKeysConstraintsVec, boundValuesConstraintsVec,
 boundKeysVariablesVec, lowerboundValuesVariablesVec, upperboundValuesVariablesVec):
 """
Parameters:

:param numConstraints: int
:param numVariables: int
:param nameVariablesVec: list
:param objLinearCoefficientsVec: list
:param objQuadraticCoefficientsMatrix: list
:param constraintsCoefficientsMatrix: list
:param boundKeysConstraintsVec: list
:param boundValuesConstraintsVec: list
:param boundKeysVariablesVec: list
:param lowerboundValuesVariablesVec: list
:param upperboundValuesVariablesVec: list

Returns:

:return: solutionsDict: dict
:return: objValue: float
"""

model =gurobipy.Model("quadratic programming") # We can set a name for model, here we set "quadratic
 # programming"

Add variables to model
variables = []
for j in xrange(numVariables):
 variables.append(
 model.addVar(
 lb=lowerboundValuesVariablesVec[j],

```

```

 ub=upperboundValuesVariablesVec[j],
 vtype=boundKeysVariablesVec[j],
 name=nameVariablesVec[j]
)
)

Populate constraints matrix
for i in xrange(numConstraints):
 expr =gurobipy.LinExpr()
 for j in xrange(numVariables):
 if (constraintsCoefficientsMatrix[i][j] !=0):
 expr +=constraintsCoefficientsMatrix[i][j] *variables[j]
 model.addConstr(
 lhs=expr,
 sense=boundKeysConstraintsVec[i],
 rhs=boundValuesConstraintsVec[i],
)

Populate objective
Add quadratic terms
objective =gurobipy.QuadExpr()
for i in xrange(numVariables):
 for j in xrange(numVariables):
 if (objQuadraticCoefficientsMatrix[i][j] !=0):
 objective +=objQuadraticCoefficientsMatrix[i][j] *variables[i] *variables[j]
Add linear terms
for j in xrange(numVariables):
 if (objLinearCoefficientsVec[j] !=0):
 objective +=objLinearCoefficientsVec[j] *variables[j]

parameter sense in function model.setObjective takes values from {gurobipy.GRB.MAXIMIZE,
gurobipy.GRB.MINIMIZE}
model.setObjective(objective, gurobipy.GRB.MINIMIZE) # Don't write out parameters' name

Solve the model
model.optimize()

Write model to a file
model.write(filename="qp.lp") # Write formulation into a file
model.write(filename="qp.sol") # Write all solutions into a file

Print solutions
if (model.status ==gurobipy.GRB.Status.OPTIMAL):
 solutionsDict ={}
 for variable_index in xrange(len(model.getVars())):
 variable =model.getVars()[variable_index]
 solutionsDict[nameVariablesVec[variable_index]] =variable.x
 objValue =model.objVal
 return solutionsDict, objValue
elif (model.status ==gurobipy.GRB.Status.INF_OR_UNBD):
 print("Model is infeasible or unbounded")
 exit(0)

```

```

 elif (model.status ==gurobipy.GRB.Status.INFEASIBLE):
 print("Model is infeasible")
 exit(0)
 elif (model.status ==gurobipy.GRB.Status.UNBOUNDED):
 print("Model is unbounded")
 exit(0)
 else:
 print("Optimization ended with status %d" % model.status)
 exit(0)

if __name__ == '__main__':
 numConstraints =2
 numVariables =3
 nameVariablesVec =["x", "y", "z"]
 objLinearCoefficientsVec =[1, 1, 0]
 objQuadraticCoefficientsMatrix =[

 [1, 1, 0],
 [0, 1, 1],
 [0, 0, 1]
]
 constraintsCoefficientsMatrix =[

 [1, 2, 3],
 [1, 1, 0]
]
 boundKeysConstraintsVec =[gurobipy.GRB.GREATER_EQUAL, gurobipy.GRB.GREATER_EQUAL]
 boundValuesConstraintsVec =[4.0, 1.0]
 boundKeysVariablesVec =[gurobipy.GRB.CONTINUOUS, gurobipy.GRB.CONTINUOUS, gurobipy.GRB.CONTINUOUS]
 lowerboundValuesVariablesVec =[0.0, 0.0, 0.0]
 upperboundValuesVariablesVec =[gurobipy.GRB.INFINITY, gurobipy.GRB.INFINITY, gurobipy.GRB.INFINITY]

 try:
 solutionsDict, objValue =qpSolver(
 numConstraints=numConstraints, numVariables=numVariables,
 nameVariablesVec=nameVariablesVec,
 objLinearCoefficientsVec=objLinearCoefficientsVec,
 objQuadraticCoefficientsMatrix=objQuadraticCoefficientsMatrix,
 constraintsCoefficientsMatrix=constraintsCoefficientsMatrix,
 boundKeysConstraintsVec=boundKeysConstraintsVec,
 boundValuesConstraintsVec=boundValuesConstraintsVec,
 boundKeysVariablesVec=boundKeysVariablesVec,
 lowerboundValuesVariablesVec=lowerboundValuesVariablesVec,
 upperboundValuesVariablesVec=upperboundValuesVariablesVec,
)
 except gurobipy.GurobiError as e:
 print('Error code ' +str(e.errno) +": " +str(e))
 except AttributeError:
 print('Encountered an attribute error')

```

### 69.3 Cplex

## 70 Graph Database

### 70.1 Neo4j