

# Deep Learning

Ian Goodfellow  
Yoshua Bengio  
Aaron Courville

# Contents

<b>Website</b>	<b>vii</b>
<b>Acknowledgments</b>	<b>viii</b>
<b>Notation</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Who Should Read This Book? . . . . .	8
1.2 Historical Trends in Deep Learning . . . . .	11
 <b>I Applied Math and Machine Learning Basics</b>	 <b>29</b>
<b>2 Linear Algebra</b>	<b>31</b>
2.1 Scalars, Vectors, Matrices and Tensors . . . . .	31
2.2 Multiplying Matrices and Vectors . . . . .	34
2.3 Identity and Inverse Matrices . . . . .	36
2.4 Linear Dependence and Span . . . . .	37
2.5 Norms . . . . .	39
2.6 Special Kinds of Matrices and Vectors . . . . .	40
2.7 Eigendecomposition . . . . .	42
2.8 Singular Value Decomposition . . . . .	44
2.9 The Moore-Penrose Pseudoinverse . . . . .	45
2.10 The Trace Operator . . . . .	46
2.11 The Determinant . . . . .	47
2.12 Example: Principal Components Analysis . . . . .	48
 <b>3 Probability and Information Theory</b>	 <b>53</b>
3.1 Why Probability? . . . . .	54

3.2	Random Variables . . . . .	56
3.3	Probability Distributions . . . . .	56
3.4	Marginal Probability . . . . .	58
3.5	Conditional Probability . . . . .	59
3.6	The Chain Rule of Conditional Probabilities . . . . .	59
3.7	Independence and Conditional Independence . . . . .	60
3.8	Expectation, Variance and Covariance . . . . .	60
3.9	Common Probability Distributions . . . . .	62
3.10	Useful Properties of Common Functions . . . . .	67
3.11	Bayes' Rule . . . . .	70
3.12	Technical Details of Continuous Variables . . . . .	71
3.13	Information Theory . . . . .	73
3.14	Structured Probabilistic Models . . . . .	75
<b>4</b>	<b>Numerical Computation</b>	<b>80</b>
4.1	Overflow and Underflow . . . . .	80
4.2	Poor Conditioning . . . . .	82
4.3	Gradient-Based Optimization . . . . .	82
4.4	Constrained Optimization . . . . .	93
4.5	Example: Linear Least Squares . . . . .	96
<b>5</b>	<b>Machine Learning Basics</b>	<b>98</b>
5.1	Learning Algorithms . . . . .	99
5.2	Capacity, Overfitting and Underfitting . . . . .	110
5.3	Hyperparameters and Validation Sets . . . . .	120
5.4	Estimators, Bias and Variance . . . . .	122
5.5	Maximum Likelihood Estimation . . . . .	131
5.6	Bayesian Statistics . . . . .	135
5.7	Supervised Learning Algorithms . . . . .	140
5.8	Unsupervised Learning Algorithms . . . . .	146
5.9	Stochastic Gradient Descent . . . . .	151
5.10	Building a Machine Learning Algorithm . . . . .	153
5.11	Challenges Motivating Deep Learning . . . . .	155
<b>II</b>	<b>Deep Networks: Modern Practices</b>	<b>166</b>
<b>6</b>	<b>Deep Feedforward Networks</b>	<b>168</b>
6.1	Example: Learning XOR . . . . .	171
6.2	Gradient-Based Learning . . . . .	177

6.3	Hidden Units . . . . .	191
6.4	Architecture Design . . . . .	197
6.5	Back-Propagation and Other Differentiation Algorithms . . . . .	204
6.6	Historical Notes . . . . .	224
<b>7</b>	<b>Regularization for Deep Learning</b>	<b>228</b>
7.1	Parameter Norm Penalties . . . . .	230
7.2	Norm Penalties as Constrained Optimization . . . . .	237
7.3	Regularization and Under-Constrained Problems . . . . .	239
7.4	Dataset Augmentation . . . . .	240
7.5	Noise Robustness . . . . .	242
7.6	Semi-Supervised Learning . . . . .	243
7.7	Multi-Task Learning . . . . .	244
7.8	Early Stopping . . . . .	246
7.9	Parameter Tying and Parameter Sharing . . . . .	253
7.10	Sparse Representations . . . . .	254
7.11	Bagging and Other Ensemble Methods . . . . .	256
7.12	Dropout . . . . .	258
7.13	Adversarial Training . . . . .	268
7.14	Tangent Distance, Tangent Prop, and Manifold Tangent Classifier	270
<b>8</b>	<b>Optimization for Training Deep Models</b>	<b>274</b>
8.1	How Learning Differs from Pure Optimization . . . . .	275
8.2	Challenges in Neural Network Optimization . . . . .	282
8.3	Basic Algorithms . . . . .	294
8.4	Parameter Initialization Strategies . . . . .	301
8.5	Algorithms with Adaptive Learning Rates . . . . .	306
8.6	Approximate Second-Order Methods . . . . .	310
8.7	Optimization Strategies and Meta-Algorithms . . . . .	317
<b>9</b>	<b>Convolutional Networks</b>	<b>330</b>
9.1	The Convolution Operation . . . . .	331
9.2	Motivation . . . . .	335
9.3	Pooling . . . . .	339
9.4	Convolution and Pooling as an Infinitely Strong Prior . . . . .	345
9.5	Variants of the Basic Convolution Function . . . . .	347
9.6	Structured Outputs . . . . .	358
9.7	Data Types . . . . .	360
9.8	Efficient Convolution Algorithms . . . . .	362
9.9	Random or Unsupervised Features . . . . .	363

9.10	The Neuroscientific Basis for Convolutional Networks . . . . .	364
9.11	Convolutional Networks and the History of Deep Learning . . . . .	371
<b>10</b>	<b>Sequence Modeling: Recurrent and Recursive Nets</b>	<b>373</b>
10.1	Unfolding Computational Graphs . . . . .	375
10.2	Recurrent Neural Networks . . . . .	378
10.3	Bidirectional RNNs . . . . .	394
10.4	Encoder-Decoder Sequence-to-Sequence Architectures . . . . .	396
10.5	Deep Recurrent Networks . . . . .	398
10.6	Recursive Neural Networks . . . . .	400
10.7	The Challenge of Long-Term Dependencies . . . . .	401
10.8	Echo State Networks . . . . .	404
10.9	Leaky Units and Other Strategies for Multiple Time Scales . . . . .	406
10.10	The Long Short-Term Memory and Other Gated RNNs . . . . .	408
10.11	Optimization for Long-Term Dependencies . . . . .	413
10.12	Explicit Memory . . . . .	416
<b>11</b>	<b>Practical Methodology</b>	<b>421</b>
11.1	Performance Metrics . . . . .	422
11.2	Default Baseline Models . . . . .	425
11.3	Determining Whether to Gather More Data . . . . .	426
11.4	Selecting Hyperparameters . . . . .	427
11.5	Debugging Strategies . . . . .	436
11.6	Example: Multi-Digit Number Recognition . . . . .	440
<b>12</b>	<b>Applications</b>	<b>443</b>
12.1	Large-Scale Deep Learning . . . . .	443
12.2	Computer Vision . . . . .	452
12.3	Speech Recognition . . . . .	458
12.4	Natural Language Processing . . . . .	461
12.5	Other Applications . . . . .	478
<b>III</b>	<b>Deep Learning Research</b>	<b>486</b>
<b>13</b>	<b>Linear Factor Models</b>	<b>489</b>
13.1	Probabilistic PCA and Factor Analysis . . . . .	490
13.2	Independent Component Analysis (ICA) . . . . .	491
13.3	Slow Feature Analysis . . . . .	493
13.4	Sparse Coding . . . . .	496

13.5	Manifold Interpretation of PCA . . . . .	499
<b>14</b>	<b>Autoencoders</b>	<b>502</b>
14.1	Undercomplete Autoencoders . . . . .	503
14.2	Regularized Autoencoders . . . . .	504
14.3	Representational Power, Layer Size and Depth . . . . .	508
14.4	Stochastic Encoders and Decoders . . . . .	509
14.5	Denoising Autoencoders . . . . .	510
14.6	Learning Manifolds with Autoencoders . . . . .	515
14.7	Contractive Autoencoders . . . . .	521
14.8	Predictive Sparse Decomposition . . . . .	523
14.9	Applications of Autoencoders . . . . .	524
<b>15</b>	<b>Representation Learning</b>	<b>526</b>
15.1	Greedy Layer-Wise Unsupervised Pretraining . . . . .	528
15.2	Transfer Learning and Domain Adaptation . . . . .	536
15.3	Semi-Supervised Disentangling of Causal Factors . . . . .	541
15.4	Distributed Representation . . . . .	546
15.5	Exponential Gains from Depth . . . . .	553
15.6	Providing Clues to Discover Underlying Causes . . . . .	554
<b>16</b>	<b>Structured Probabilistic Models for Deep Learning</b>	<b>558</b>
16.1	The Challenge of Unstructured Modeling . . . . .	559
16.2	Using Graphs to Describe Model Structure . . . . .	563
16.3	Sampling from Graphical Models . . . . .	580
16.4	Advantages of Structured Modeling . . . . .	582
16.5	Learning about Dependencies . . . . .	582
16.6	Inference and Approximate Inference . . . . .	584
16.7	The Deep Learning Approach to Structured Probabilistic Models	585
<b>17</b>	<b>Monte Carlo Methods</b>	<b>590</b>
17.1	Sampling and Monte Carlo Methods . . . . .	590
17.2	Importance Sampling . . . . .	592
17.3	Markov Chain Monte Carlo Methods . . . . .	595
17.4	Gibbs Sampling . . . . .	599
17.5	The Challenge of Mixing between Separated Modes . . . . .	599
<b>18</b>	<b>Confronting the Partition Function</b>	<b>605</b>
18.1	The Log-Likelihood Gradient . . . . .	606
18.2	Stochastic Maximum Likelihood and Contrastive Divergence . . .	607

18.3	Pseudolikelihood . . . . .	615
18.4	Score Matching and Ratio Matching . . . . .	617
18.5	Denoising Score Matching . . . . .	619
18.6	Noise-Contrastive Estimation . . . . .	620
18.7	Estimating the Partition Function . . . . .	623
<b>19</b>	<b>Approximate Inference</b>	<b>631</b>
19.1	Inference as Optimization . . . . .	633
19.2	Expectation Maximization . . . . .	634
19.3	MAP Inference and Sparse Coding . . . . .	635
19.4	Variational Inference and Learning . . . . .	638
19.5	Learned Approximate Inference . . . . .	651
<b>20</b>	<b>Deep Generative Models</b>	<b>654</b>
20.1	Boltzmann Machines . . . . .	654
20.2	Restricted Boltzmann Machines . . . . .	656
20.3	Deep Belief Networks . . . . .	660
20.4	Deep Boltzmann Machines . . . . .	663
20.5	Boltzmann Machines for Real-Valued Data . . . . .	676
20.6	Convolutional Boltzmann Machines . . . . .	683
20.7	Boltzmann Machines for Structured or Sequential Outputs . . . . .	685
20.8	Other Boltzmann Machines . . . . .	686
20.9	Back-Propagation through Random Operations . . . . .	687
20.10	Directed Generative Nets . . . . .	692
20.11	Drawing Samples from Autoencoders . . . . .	711
20.12	Generative Stochastic Networks . . . . .	714
20.13	Other Generation Schemes . . . . .	716
20.14	Evaluating Generative Models . . . . .	717
20.15	Conclusion . . . . .	720
	<b>Bibliography</b>	<b>721</b>
	<b>Index</b>	<b>777</b>

# Website

[www.deeplearningbook.org](http://www.deeplearningbook.org)

This book is accompanied by the above website. The website provides a variety of supplementary material, including exercises, lecture slides, corrections of mistakes, and other resources that should be useful to both readers and instructors.



# Acknowledgments

This book would not have been possible without the contributions of many people.

We would like to thank those who commented on our proposal for the book and helped plan its contents and organization: Guillaume Alain, Kyunghyun Cho, Çağlar Gülçehre, David Krueger, Hugo Larochelle, Razvan Pascanu and Thomas Rohée.

We would like to thank the people who offered feedback on the content of the book itself. Some offered feedback on many chapters: Martín Abadi, Guillaume Alain, Ion Androutsopoulos, Fred Bertsch, Olexa Bilaniuk, Ufuk Can Biçici, Matko Bošnjak, John Boersma, Greg Brockman, Alexandre de Brébisson, Pierre Luc Carrier, Sarath Chandar, Pawel Chilinski, Mark Daoust, Oleg Dashevskii, Laurent Dinh, Stephan Dreseidl, Jim Fan, Miao Fan, Meire Fortunato, Frédéric Francis, Nando de Freitas, Çağlar Gülçehre, Jurgen Van Gael, Javier Alonso García, Jonathan Hunt, Gopi Jeyaram, Chingiz Kabayev, Lukasz Kaiser, Varun Kanade, Asifullah Khan, Akiel Khan, John King, Diederik P. Kingma, Yann LeCun, Rudolf Mathey, Matías Mattamala, Abhinav Maurya, Kevin Murphy, Oleg Mürk, Roman Novak, Augustus Q. Odena, Simon Pavlik, Karl Pichotta, Eddie Pierce, Kari Pulli, Roussel Rahman, Tapani Raiko, Anurag Ranjan, Johannes Roith, Mihaela Rosca, Halis Sak, César Salgado, Grigory Sapunov, Yoshinori Sasaki, Mike Schuster, Julian Serban, Nir Shabat, Ken Shirriff, Andre Simpelo, Scott Stanley, David Sussillo, Ilya Sutskever, Carles Gelada Sáez, Graham Taylor, Valentin Tolmer, Massimiliano Tomassoli, An Tran, Shubhendu Trivedi, Alexey Umnov, Vincent Vanhoucke, Marco Visentini-Scarzanella, Martin Vita, David Warde-Farley, Dustin Webb, Kelvin Xu, Wei Xue, Ke Yang, Li Yao, Zygmunt Zajac and Ozan Çağlayan.

We would also like to thank those who provided us with useful feedback on individual chapters:

- **Notation:** Zhang Yuanhang.
- Chapter 1, **Introduction:** Yusuf Akgul, Sebastien Bratieres, Samira Ebrahimi,

Charlie Gorichanaz, Brendan Loudermilk, Eric Morris, Cosmin Pârvulescu and Alfredo Solano.

- Chapter 2, **Linear Algebra**: Amjad Almahairi, Nikola Banić, Kevin Bennett, Philippe Castonguay, Oscar Chang, Eric Fosler-Lussier, Andrey Khalyavin, Sergey Oreshkov, István Petrás, Dennis Prangle, Thomas Rohée, Gitanjali Gulve Sehgal, Colby Toland, Alessandro Vitale and Bob Welland.
- Chapter 3, **Probability and Information Theory**: John Philip Anderson, Kai Arulkumaran, Vincent Dumoulin, Rui Fa, Stephan Gouws, Artem Oboturov, Antti Rasmus, Alexey Surkov and Volker Tresp.
- Chapter 4, **Numerical Computation**: Tran Lam An Ian Fischer and Hu Yuhuang.
- Chapter 5, **Machine Learning Basics**: Dzmitry Bahdanau, Justin Domingue, Nikhil Garg, Makoto Otsuka, Bob Pepin, Philip Popien, Emmanuel Rayner, Peter Shepard, Kee-Bong Song, Zheng Sun and Andy Wu.
- Chapter 6, **Deep Feedforward Networks**: Uriel Berdugo, Fabrizio Bottarel, Elizabeth Burl, Ishan Durugkar, Jeff Hlywa, Jong Wook Kim, David Krueger and Aditya Kumar Praharaaj.
- Chapter 7, **Regularization for Deep Learning**: Morten Kolbæk, Kshitij Lauria, Inkyu Lee, Sunil Mohan, Hai Phong Phan and Joshua Salisbury.
- Chapter 8, **Optimization for Training Deep Models**: Marcel Ackermann, Peter Armitage, Rowel Atienza, Andrew Brock, Tegan Maharaj, James Martens, Kashif Rasul, Klaus Strobl and Nicholas Turner.
- Chapter 9, **Convolutional Networks**: Martín Arjovsky, Eugene Brevdo, Konstantin Divilov, Eric Jensen, Mehdi Mirza, Alex Paino, Marjorie Sayer, Ryan Stout and Wentao Wu.
- Chapter 10, **Sequence Modeling: Recurrent and Recursive Nets**: Gökçen Eraslan, Steven Hickson, Razvan Pascanu, Lorenzo von Ritter, Rui Rodrigues, Dmitriy Serdyuk, Dongyu Shi and Kaiyu Yang.
- Chapter 11, **Practical Methodology**: Daniel Beckstein.
- Chapter 12, **Applications**: George Dahl, Vladimir Nekrasov and Ribana Roscher.
- Chapter 13, **Linear Factor Models**: Jayanth Koushik.

- Chapter 15, **Representation Learning**: Kunal Ghosh.
- Chapter 16, **Structured Probabilistic Models for Deep Learning**: Minh Lê and Anton Varfolom.
- Chapter 18, **Confronting the Partition Function**: Sam Bowman.
- Chapter 19, **Approximate Inference**: Yujia Bao.
- Chapter 20, **Deep Generative Models**: Nicolas Chapados, Daniel Galvez, Wenming Ma, Fady Medhat, Shakir Mohamed and Grégoire Montavon.
- Bibliography: Lukas Michelbacher and Leslie N. Smith.

We also want to thank those who allowed us to reproduce images, figures or data from their publications. We indicate their contributions in the figure captions throughout the text.

We would like to thank Lu Wang for writing pdf2htmlEX, which we used to make the web version of the book, and for offering support to improve the quality of the resulting HTML.

We would like to thank Ian's wife Daniela Flori Goodfellow for patiently supporting Ian during the writing of the book as well as for help with proofreading.

We would like to thank the Google Brain team for providing an intellectual environment where Ian could devote a tremendous amount of time to writing this book and receive feedback and guidance from colleagues. We would especially like to thank Ian's former manager, Greg Corrado, and his current manager, Samy Bengio, for their support of this project. Finally, we would like to thank Geoffrey Hinton for encouragement when writing was difficult.

# Notation

This section provides a concise reference describing the notation used throughout this book. If you are unfamiliar with any of the corresponding mathematical concepts, we describe most of these ideas in chapters 2–4.

## Numbers and Arrays

$a$	A scalar (integer or real)
$\mathbf{a}$	A vector
$\mathbf{A}$	A matrix
$\mathbf{A}$	A tensor
$\mathbf{I}_n$	Identity matrix with $n$ rows and $n$ columns
$\mathbf{I}$	Identity matrix with dimensionality implied by context
$\mathbf{e}^{(i)}$	Standard basis vector $[0, \dots, 0, 1, 0, \dots, 0]$ with a 1 at position $i$
$\text{diag}(\mathbf{a})$	A square, diagonal matrix with diagonal entries given by $\mathbf{a}$
$a$	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

## Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and $n$
$[a, b]$	The real interval including $a$ and $b$
$(a, b]$	The real interval excluding $a$ but including $b$
$\mathbb{A} \setminus \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\mathcal{G}$	A graph
$Pa_{\mathcal{G}}(x_i)$	The parents of $x_i$ in $\mathcal{G}$

## Indexing

$a_i$	Element $i$ of vector $\mathbf{a}$ , with indexing starting at 1
$a_{-i}$	All elements of vector $\mathbf{a}$ except for element $i$
$A_{i,j}$	Element $i, j$ of matrix $\mathbf{A}$
$\mathbf{A}_{i,:}$	Row $i$ of matrix $\mathbf{A}$
$\mathbf{A}_{:,i}$	Column $i$ of matrix $\mathbf{A}$
$A_{i,j,k}$	Element $(i, j, k)$ of a 3-D tensor $\mathbf{A}$
$\mathbf{A}_{::,i}$	2-D slice of a 3-D tensor
$a_i$	Element $i$ of the random vector $\mathbf{a}$

## Linear Algebra Operations

$\mathbf{A}^{\top}$	Transpose of matrix $\mathbf{A}$
$\mathbf{A}^+$	Moore-Penrose pseudoinverse of $\mathbf{A}$
$\mathbf{A} \odot \mathbf{B}$	Element-wise (Hadamard) product of $\mathbf{A}$ and $\mathbf{B}$
$\det(\mathbf{A})$	Determinant of $\mathbf{A}$

## Calculus

$\frac{dy}{dx}$	Derivative of $y$ with respect to $x$
$\frac{\partial y}{\partial x}$	Partial derivative of $y$ with respect to $x$
$\nabla_{\mathbf{x}} y$	Gradient of $y$ with respect to $\mathbf{x}$
$\nabla_{\mathbf{X}} y$	Matrix derivatives of $y$ with respect to $\mathbf{X}$
$\nabla_{\mathbf{x}} y$	Tensor containing derivatives of $y$ with respect to $\mathbf{X}$
$\frac{\partial f}{\partial \mathbf{x}}$	Jacobian matrix $\mathbf{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$
$\nabla_{\mathbf{x}}^2 f(\mathbf{x})$ or $\mathbf{H}(f)(\mathbf{x})$	The Hessian matrix of $f$ at input point $\mathbf{x}$
$\int f(\mathbf{x}) d\mathbf{x}$	Definite integral over the entire domain of $\mathbf{x}$
$\int_{\mathbb{S}} f(\mathbf{x}) d\mathbf{x}$	Definite integral with respect to $\mathbf{x}$ over the set $\mathbb{S}$

## Probability and Information Theory

$a \perp b$	The random variables $a$ and $b$ are independent
$a \perp b \mid c$	They are conditionally independent given $c$
$P(a)$	A probability distribution over a discrete variable
$p(a)$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$a \sim P$	Random variable $a$ has distribution $P$
$\mathbb{E}_{\mathbf{x} \sim P}[f(\mathbf{x})]$ or $\mathbb{E}f(\mathbf{x})$	Expectation of $f(\mathbf{x})$ with respect to $P(\mathbf{x})$
$\text{Var}(f(\mathbf{x}))$	Variance of $f(\mathbf{x})$ under $P(\mathbf{x})$
$\text{Cov}(f(\mathbf{x}), g(\mathbf{x}))$	Covariance of $f(\mathbf{x})$ and $g(\mathbf{x})$ under $P(\mathbf{x})$
$H(\mathbf{x})$	Shannon entropy of the random variable $\mathbf{x}$
$D_{\text{KL}}(P \parallel Q)$	Kullback-Leibler divergence of $P$ and $Q$
$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\mathbf{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

### Functions

$f : \mathbb{A} \rightarrow \mathbb{B}$	The function $f$ with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions $f$ and $g$
$f(\mathbf{x}; \boldsymbol{\theta})$	A function of $\mathbf{x}$ parametrized by $\boldsymbol{\theta}$ . (Sometimes we write $f(\mathbf{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of $x$
$\sigma(x)$	Logistic sigmoid, $\frac{1}{1 + \exp(-x)}$
$\zeta(x)$	Softplus, $\log(1 + \exp(x))$
$\ \mathbf{x}\ _p$	$L^p$ norm of $\mathbf{x}$
$\ \mathbf{x}\ $	$L^2$ norm of $\mathbf{x}$
$x^+$	Positive part of $x$ , i.e., $\max(0, x)$
$\mathbf{1}_{\text{condition}}$	is 1 if the condition is true, 0 otherwise

Sometimes we use a function  $f$  whose argument is a scalar but apply it to a vector, matrix, or tensor:  $f(\mathbf{x})$ ,  $f(\mathbf{X})$ , or  $f(\mathbf{X})$ . This denotes the application of  $f$  to the array element-wise. For example, if  $\mathbf{C} = \sigma(\mathbf{X})$ , then  $C_{i,j,k} = \sigma(X_{i,j,k})$  for all valid values of  $i$ ,  $j$  and  $k$ .

### Datasets and Distributions

$p_{\text{data}}$	The data generating distribution
$\hat{p}_{\text{data}}$	The empirical distribution defined by the training set
$\mathbb{X}$	A set of training examples
$\mathbf{x}^{(i)}$	The $i$ -th example (input) from a dataset
$y^{(i)}$ or $\mathbf{y}^{(i)}$	The target associated with $\mathbf{x}^{(i)}$ for supervised learning
$\mathbf{X}$	The $m \times n$ matrix with input example $\mathbf{x}^{(i)}$ in row $\mathbf{X}_{i,:}$

# Chapter 1

## Introduction

Inventors have long dreamed of creating machines that think. This desire dates back to at least the time of ancient Greece. The mythical figures Pygmalion, Daedalus, and Hephaestus may all be interpreted as legendary inventors, and Galatea, Talos, and Pandora may all be regarded as artificial life ([Ovid and Martin, 2004](#); [Sparkes, 1996](#); [Tandy, 1997](#)).

When programmable computers were first conceived, people wondered whether such machines might become intelligent, over a hundred years before one was built ([Lovelace, 1842](#)). Today, **artificial intelligence** (AI) is a thriving field with many practical applications and active research topics. We look to intelligent software to automate routine labor, understand speech or images, make diagnoses in medicine and support basic scientific research.

In the early days of artificial intelligence, the field rapidly tackled and solved problems that are intellectually difficult for human beings but relatively straightforward for computers—problems that can be described by a list of formal, mathematical rules. The true challenge to artificial intelligence proved to be solving the tasks that are easy for people to perform but hard for people to describe formally—problems that we solve intuitively, that feel automatic, like recognizing spoken words or faces in images.

This book is about a solution to these more intuitive problems. This solution is to allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined in terms of its relation to simpler concepts. By gathering knowledge from experience, this approach avoids the need for human operators to formally specify all of the knowledge that the computer needs. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these



concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI **deep learning**.

Many of the early successes of AI took place in relatively sterile and formal environments and did not require computers to have much knowledge about the world. For example, IBM's Deep Blue chess-playing system defeated world champion Garry Kasparov in 1997 (Hsu, 2002). Chess is of course a very simple world, containing only sixty-four locations and thirty-two pieces that can move in only rigidly circumscribed ways. Devising a successful chess strategy is a tremendous accomplishment, but the challenge is not due to the difficulty of describing the set of chess pieces and allowable moves to the computer. Chess can be completely described by a very brief list of completely formal rules, easily provided ahead of time by the programmer.

Ironically, abstract and formal tasks that are among the most difficult mental undertakings for a human being are among the easiest for a computer. Computers have long been able to defeat even the best human chess player, but are only recently matching some of the abilities of average human beings to recognize objects or speech. A person's everyday life requires an immense amount of knowledge about the world. Much of this knowledge is subjective and intuitive, and therefore difficult to articulate in a formal way. Computers need to capture this same knowledge in order to behave in an intelligent way. One of the key challenges in artificial intelligence is how to get this informal knowledge into a computer.

Several artificial intelligence projects have sought to hard-code knowledge about the world in formal languages. A computer can reason about statements in these formal languages automatically using logical inference rules. This is known as the **knowledge base** approach to artificial intelligence. None of these projects has led to a major success. One of the most famous such projects is Cyc (Lenat and Guha, 1989). Cyc is an inference engine and a database of statements in a language called CycL. These statements are entered by a staff of human supervisors. It is an unwieldy process. People struggle to devise formal rules with enough complexity to accurately describe the world. For example, Cyc failed to understand a story about a person named Fred shaving in the morning (Linde, 1992). Its inference engine detected an inconsistency in the story: it knew that people do not have electrical parts, but because Fred was holding an electric razor, it believed the entity "FredWhileShaving" contained electrical parts. It therefore asked whether Fred was still a person while he was shaving.

The difficulties faced by systems relying on hard-coded knowledge suggest that AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data. This capability is known as **machine learning**. The

introduction of machine learning allowed computers to tackle problems involving knowledge of the real world and make decisions that appear subjective. A simple machine learning algorithm called **logistic regression** can determine whether to recommend cesarean delivery (Mor-Yosef *et al.*, 1990). A simple machine learning algorithm called **naive Bayes** can separate legitimate e-mail from spam e-mail.

The performance of these simple machine learning algorithms depends heavily on the **representation** of the data they are given. For example, when logistic regression is used to recommend cesarean delivery, the AI system does not examine the patient directly. Instead, the doctor tells the system several pieces of relevant information, such as the presence or absence of a uterine scar. Each piece of information included in the representation of the patient is known as a **feature**. Logistic regression learns how each of these features of the patient correlates with various outcomes. However, it cannot influence the way that the features are defined in any way. If logistic regression was given an MRI scan of the patient, rather than the doctor’s formalized report, it would not be able to make useful predictions. Individual pixels in an MRI scan have negligible correlation with any complications that might occur during delivery.

This dependence on representations is a general phenomenon that appears throughout computer science and even daily life. In computer science, operations such as searching a collection of data can proceed exponentially faster if the collection is structured and indexed intelligently. People can easily perform arithmetic on Arabic numerals, but find arithmetic on Roman numerals much more time-consuming. It is not surprising that the choice of representation has an enormous effect on the performance of machine learning algorithms. For a simple visual example, see figure 1.1.

Many artificial intelligence tasks can be solved by designing the right set of features to extract for that task, then providing these features to a simple machine learning algorithm. For example, a useful feature for speaker identification from sound is an estimate of the size of speaker’s vocal tract. It therefore gives a strong clue as to whether the speaker is a man, woman, or child.

However, for many tasks, it is difficult to know what features should be extracted. For example, suppose that we would like to write a program to detect cars in photographs. We know that cars have wheels, so we might like to use the presence of a wheel as a feature. Unfortunately, it is difficult to describe exactly what a wheel looks like in terms of pixel values. A wheel has a simple geometric shape but its image may be complicated by shadows falling on the wheel, the sun glaring off the metal parts of the wheel, the fender of the car or an object in the foreground obscuring part of the wheel, and so on.

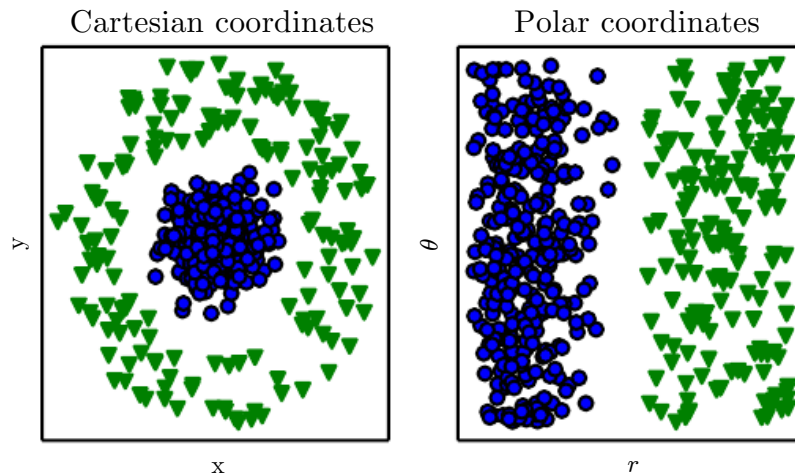


Figure 1.1: Example of different representations: suppose we want to separate two categories of data by drawing a line between them in a scatterplot. In the plot on the left, we represent some data using Cartesian coordinates, and the task is impossible. In the plot on the right, we represent the data with polar coordinates and the task becomes simple to solve with a vertical line. Figure produced in collaboration with David Warde-Farley.

One solution to this problem is to use machine learning to discover not only the mapping from representation to output but also the representation itself. This approach is known as **representation learning**. Learned representations often result in much better performance than can be obtained with hand-designed representations. They also allow AI systems to rapidly adapt to new tasks, with minimal human intervention. A representation learning algorithm can discover a good set of features for a simple task in minutes, or a complex task in hours to months. Manually designing features for a complex task requires a great deal of human time and effort; it can take decades for an entire community of researchers.

The quintessential example of a representation learning algorithm is the **autoencoder**. An autoencoder is the combination of an **encoder** function that converts the input data into a different representation, and a **decoder** function that converts the new representation back into the original format. Autoencoders are trained to preserve as much information as possible when an input is run through the encoder and then the decoder, but are also trained to make the new representation have various nice properties. Different kinds of autoencoders aim to achieve different kinds of properties.

When designing features or algorithms for learning features, our goal is usually to separate the **factors of variation** that explain the observed data. In this context, we use the word “factors” simply to refer to separate sources of influence; the factors are usually not combined by multiplication. Such factors are often not

quantities that are directly observed. Instead, they may exist either as unobserved objects or unobserved forces in the physical world that affect observable quantities. They may also exist as constructs in the human mind that provide useful simplifying explanations or inferred causes of the observed data. They can be thought of as concepts or abstractions that help us make sense of the rich variability in the data. When analyzing a speech recording, the factors of variation include the speaker's age, their sex, their accent and the words that they are speaking. When analyzing an image of a car, the factors of variation include the position of the car, its color, and the angle and brightness of the sun.

A major source of difficulty in many real-world artificial intelligence applications is that many of the factors of variation influence every single piece of data we are able to observe. The individual pixels in an image of a red car might be very close to black at night. The shape of the car's silhouette depends on the viewing angle. Most applications require us to *disentangle* the factors of variation and discard the ones that we do not care about.

Of course, it can be very difficult to extract such high-level, abstract features from raw data. Many of these factors of variation, such as a speaker's accent, can be identified only using sophisticated, nearly human-level understanding of the data. When it is nearly as difficult to obtain a representation as to solve the original problem, representation learning does not, at first glance, seem to help us.

**Deep learning** solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations. Deep learning allows the computer to build complex concepts out of simpler concepts. Figure 1.2 shows how a deep learning system can represent the concept of an image of a person by combining simpler concepts, such as corners and contours, which are in turn defined in terms of edges.

The quintessential example of a deep learning model is the feedforward deep network or **multilayer perceptron** (MLP). A multilayer perceptron is just a mathematical function mapping some set of input values to output values. The function is formed by composing many simpler functions. We can think of each application of a different mathematical function as providing a new representation of the input.

The idea of learning the right representation for the data provides one perspective on deep learning. Another perspective on deep learning is that depth allows the computer to learn a multi-step computer program. Each layer of the representation can be thought of as the state of the computer's memory after executing another set of instructions in parallel. Networks with greater depth can execute more instructions in sequence. Sequential instructions offer great power because later

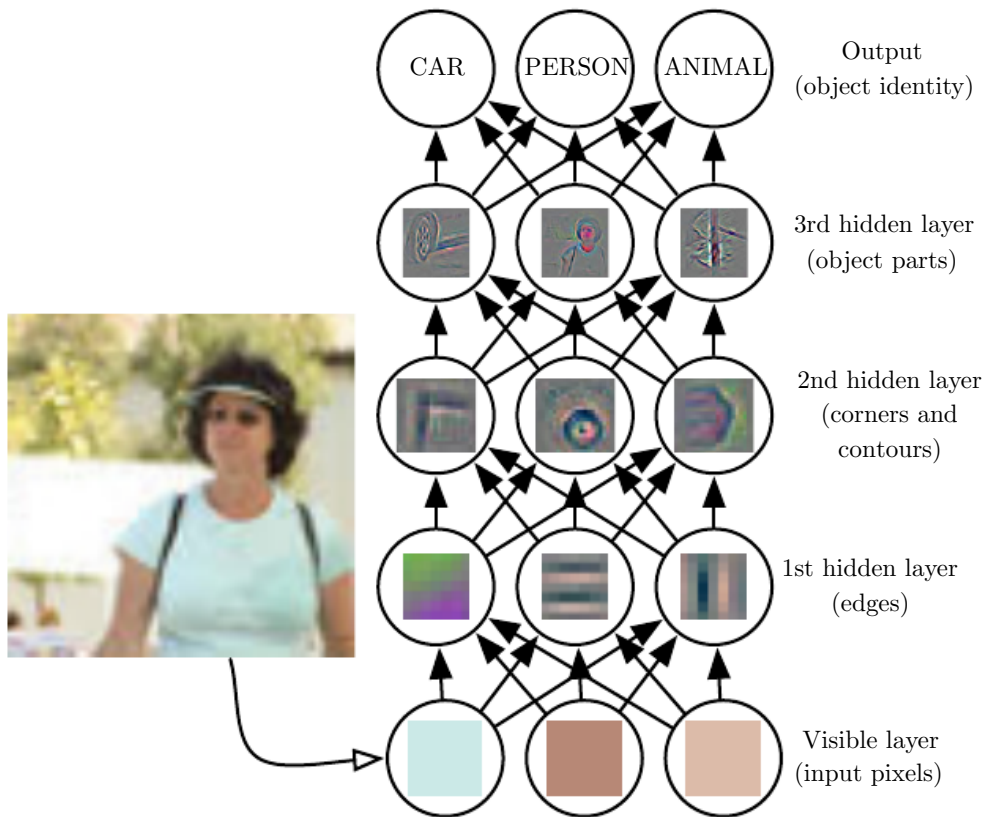


Figure 1.2: Illustration of a deep learning model. It is difficult for a computer to understand the meaning of raw sensory input data, such as this image represented as a collection of pixel values. The function mapping from a set of pixels to an object identity is very complicated. Learning or evaluating this mapping seems insurmountable if tackled directly. Deep learning resolves this difficulty by breaking the desired complicated mapping into a series of nested simple mappings, each described by a different layer of the model. The input is presented at the **visible layer**, so named because it contains the variables that we are able to observe. Then a series of **hidden layers** extracts increasingly abstract features from the image. These layers are called “hidden” because their values are not given in the data; instead the model must determine which concepts are useful for explaining the relationships in the observed data. The images here are visualizations of the kind of feature represented by each hidden unit. Given the pixels, the first layer can easily identify edges, by comparing the brightness of neighboring pixels. Given the first hidden layer’s description of the edges, the second hidden layer can easily search for corners and extended contours, which are recognizable as collections of edges. Given the second hidden layer’s description of the image in terms of corners and contours, the third hidden layer can detect entire parts of specific objects, by finding specific collections of contours and corners. Finally, this description of the image in terms of the object parts it contains can be used to recognize the objects present in the image. Images reproduced with permission from [Zeiler and Fergus \(2014\)](#).

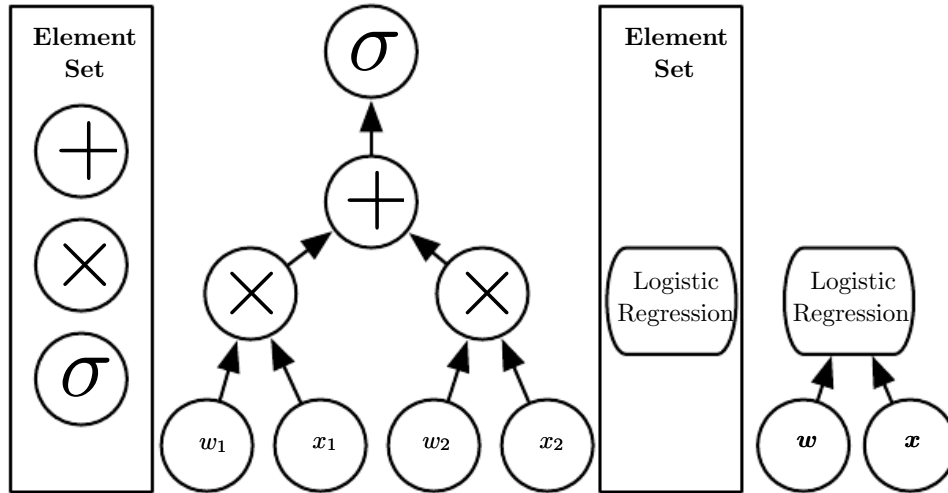


Figure 1.3: Illustration of computational graphs mapping an input to an output where each node performs an operation. Depth is the length of the longest path from input to output but depends on the definition of what constitutes a possible computational step. The computation depicted in these graphs is the output of a logistic regression model,  $\sigma(\mathbf{w}^T \mathbf{x})$ , where  $\sigma$  is the logistic sigmoid function. If we use addition, multiplication and logistic sigmoids as the elements of our computer language, then this model has depth three. If we view logistic regression as an element itself, then this model has depth one.

instructions can refer back to the results of earlier instructions. According to this view of deep learning, not all of the information in a layer's activations necessarily encodes factors of variation that explain the input. The representation also stores state information that helps to execute a program that can make sense of the input. This state information could be analogous to a counter or pointer in a traditional computer program. It has nothing to do with the content of the input specifically, but it helps the model to organize its processing.

There are two main ways of measuring the depth of a model. The first view is based on the number of sequential instructions that must be executed to evaluate the architecture. We can think of this as the length of the longest path through a flow chart that describes how to compute each of the model's outputs given its inputs. Just as two equivalent computer programs will have different lengths depending on which language the program is written in, the same function may be drawn as a flowchart with different depths depending on which functions we allow to be used as individual steps in the flowchart. Figure 1.3 illustrates how this choice of language can give two different measurements for the same architecture.

Another approach, used by deep probabilistic models, regards the depth of a model as being not the depth of the computational graph but the depth of the graph describing how concepts are related to each other. In this case, the depth



of the flowchart of the computations needed to compute the representation of each concept may be much deeper than the graph of the concepts themselves. This is because the system’s understanding of the simpler concepts can be refined given information about the more complex concepts. For example, an AI system observing an image of a face with one eye in shadow may initially only see one eye. After detecting that a face is present, it can then infer that a second eye is probably present as well. In this case, the graph of concepts only includes two layers—a layer for eyes and a layer for faces—but the graph of computations includes  $2n$  layers if we refine our estimate of each concept given the other  $n$  times.

Because it is not always clear which of these two views—the depth of the computational graph, or the depth of the probabilistic modeling graph—is most relevant, and because different people choose different sets of smallest elements from which to construct their graphs, there is no single correct value for the depth of an architecture, just as there is no single correct value for the length of a computer program. Nor is there a consensus about how much depth a model requires to qualify as “deep.” However, deep learning can safely be regarded as the study of models that either involve a greater amount of composition of learned functions or learned concepts than traditional machine learning does.

To summarize, deep learning, the subject of this book, is an approach to AI. Specifically, it is a type of machine learning, a technique that allows computer systems to improve with experience and data. According to the authors of this book, machine learning is the only viable approach to building AI systems that can operate in complicated, real-world environments. Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. Figure 1.4 illustrates the relationship between these different AI disciplines. Figure 1.5 gives a high-level schematic of how each works.

## 1.1 Who Should Read This Book?

This book can be useful for a variety of readers, but we wrote it with two main target audiences in mind. One of these target audiences is university students (undergraduate or graduate) learning about machine learning, including those who are beginning a career in deep learning and artificial intelligence research. The other target audience is software engineers who do not have a machine learning or statistics background, but want to rapidly acquire one and begin using deep learning in their product or platform. Deep learning has already proven useful in

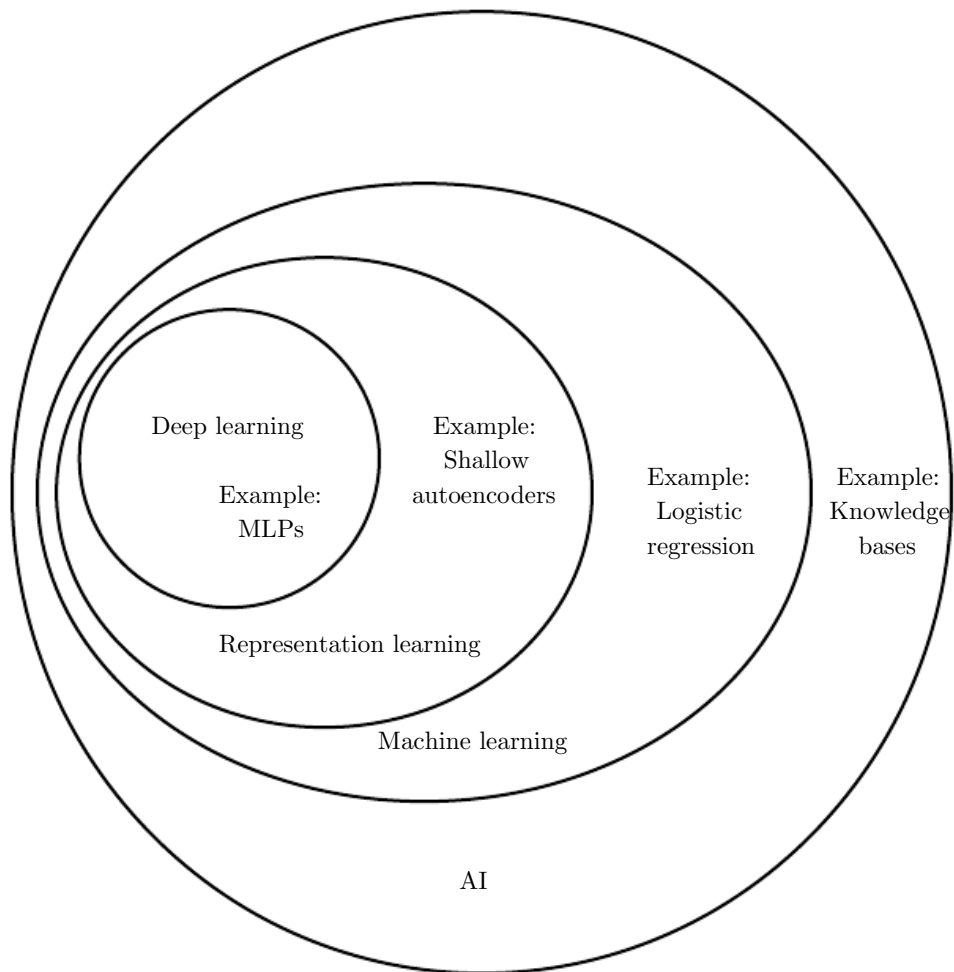


Figure 1.4: A Venn diagram showing how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is used for many but not all approaches to AI. Each section of the Venn diagram includes an example of an AI technology.



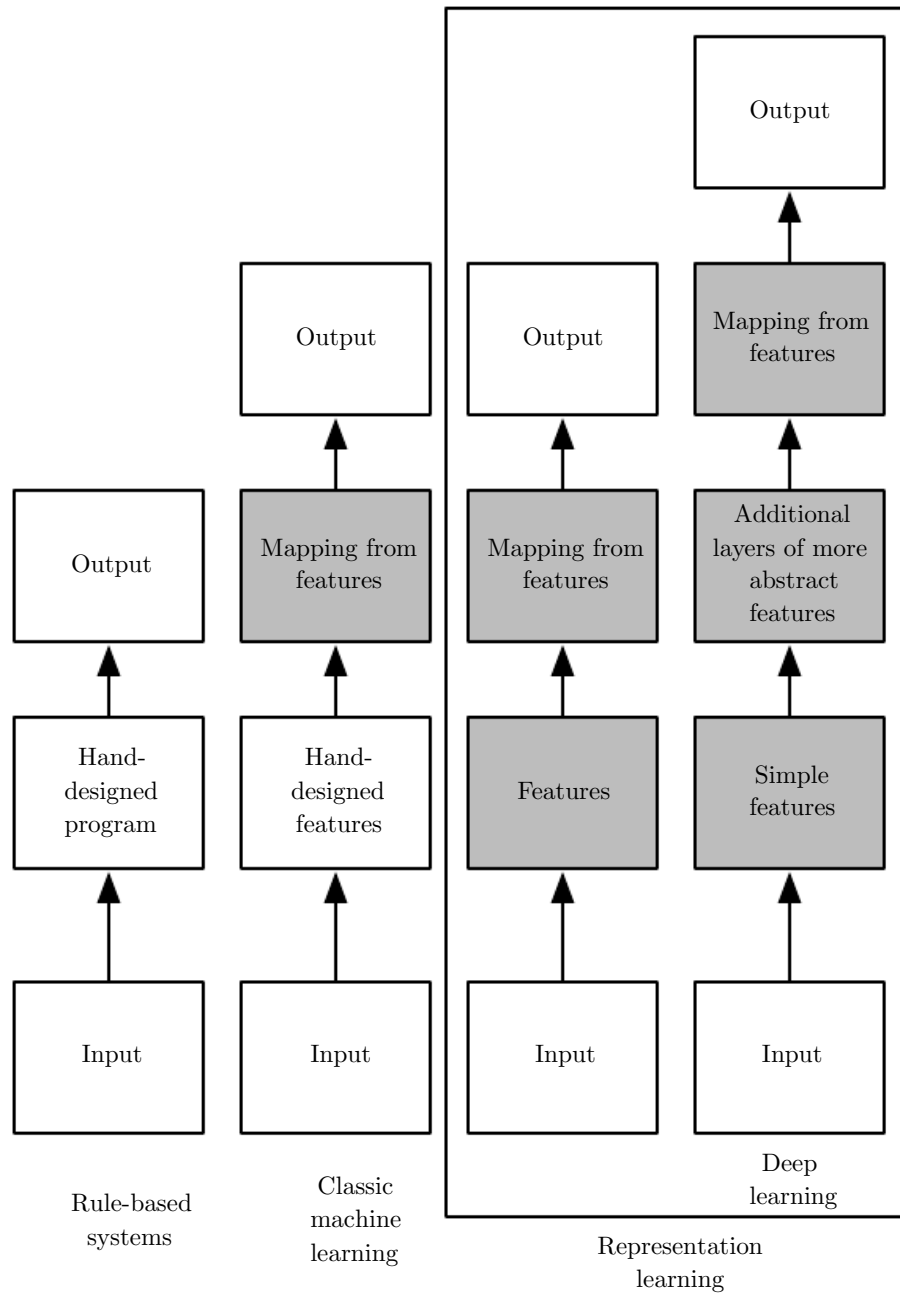


Figure 1.5: Flowcharts showing how the different parts of an AI system relate to each other within different AI disciplines. Shaded boxes indicate components that are able to learn from data.

many software disciplines including computer vision, speech and audio processing, natural language processing, robotics, bioinformatics and chemistry, video games, search engines, online advertising and finance.

This book has been organized into three parts in order to best accommodate a variety of readers. Part **I** introduces basic mathematical tools and machine learning concepts. Part **II** describes the most established deep learning algorithms that are essentially solved technologies. Part **III** describes more speculative ideas that are widely believed to be important for future research in deep learning.

Readers should feel free to skip parts that are not relevant given their interests or background. Readers familiar with linear algebra, probability, and fundamental machine learning concepts can skip part **I**, for example, while readers who just want to implement a working system need not read beyond part **II**. To help choose which chapters to read, figure 1.6 provides a flowchart showing the high-level organization of the book.

We do assume that all readers come from a computer science background. We assume familiarity with programming, a basic understanding of computational performance issues, complexity theory, introductory level calculus and some of the terminology of graph theory.

## 1.2 Historical Trends in Deep Learning

It is easiest to understand deep learning with some historical context. Rather than providing a detailed history of deep learning, we identify a few key trends:

- Deep learning has had a long and rich history, but has gone by many names reflecting different philosophical viewpoints, and has waxed and waned in popularity.
- Deep learning has become more useful as the amount of available training data has increased.
- Deep learning models have grown in size over time as computer infrastructure (both hardware and software) for deep learning has improved.
- Deep learning has solved increasingly complicated applications with increasing accuracy over time.

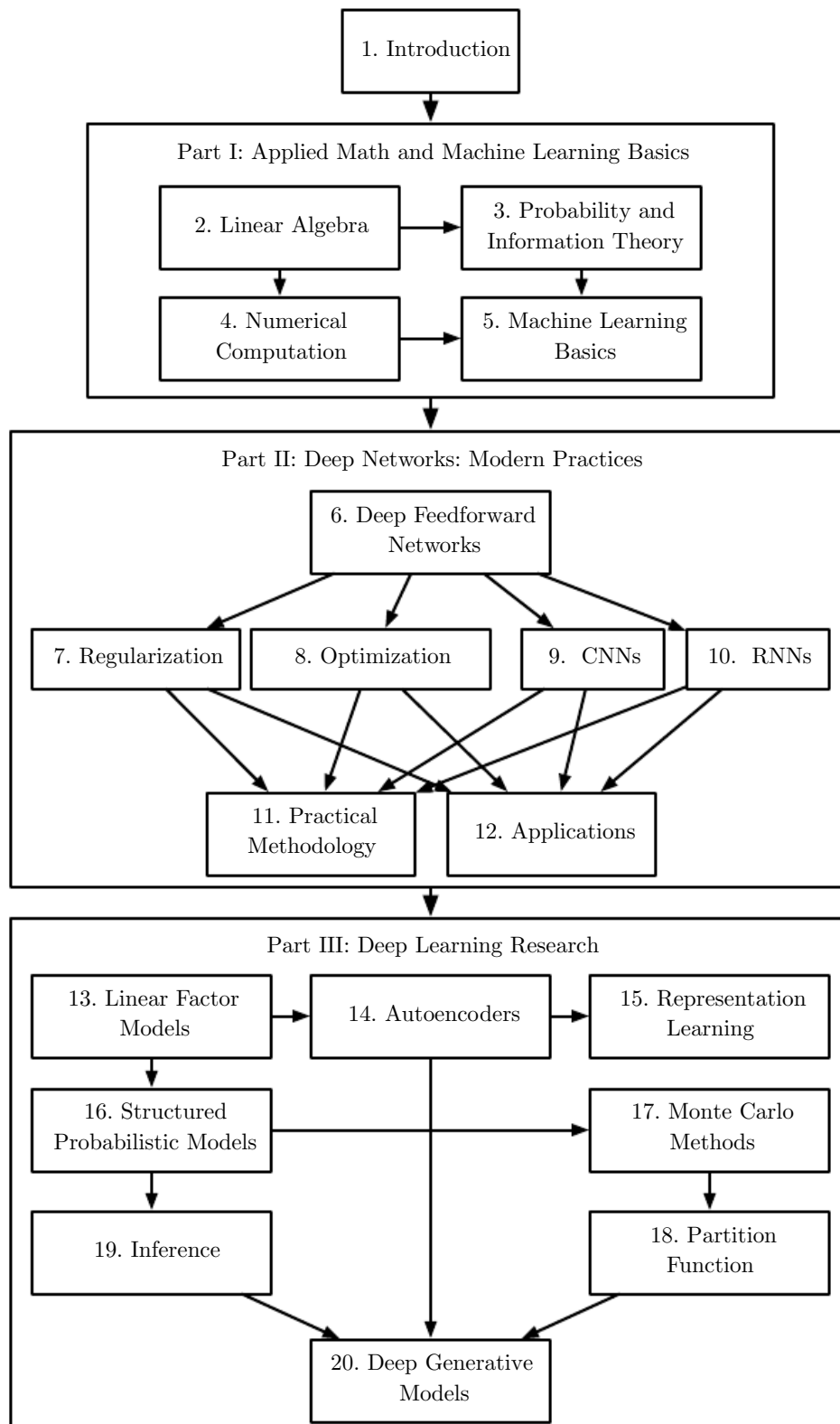


Figure 1.6: The high-level organization of the book. An arrow from one chapter to another indicates that the former chapter is prerequisite material for understanding the latter.

### 1.2.1 The Many Names and Changing Fortunes of Neural Networks

We expect that many readers of this book have heard of deep learning as an exciting new technology, and are surprised to see a mention of “history” in a book about an emerging field. In fact, deep learning dates back to the 1940s. Deep learning only *appears* to be new, because it was relatively unpopular for several years preceding its current popularity, and because it has gone through many different names, and has only recently become called “deep learning.” The field has been rebranded many times, reflecting the influence of different researchers and different perspectives.

A comprehensive history of deep learning is beyond the scope of this textbook. However, some basic context is useful for understanding deep learning. Broadly speaking, there have been three waves of development of deep learning: deep learning known as **cybernetics** in the 1940s–1960s, deep learning known as **connectionism** in the 1980s–1990s, and the current resurgence under the name deep learning beginning in 2006. This is quantitatively illustrated in figure 1.7.

Some of the earliest learning algorithms we recognize today were intended to be computational models of biological learning, i.e. models of how learning happens or could happen in the brain. As a result, one of the names that deep learning has gone by is **artificial neural networks** (ANNs). The corresponding perspective on deep learning models is that they are engineered systems inspired by the biological brain (whether the human brain or the brain of another animal). While the kinds of neural networks used for machine learning have sometimes been used to understand brain function ([Hinton and Shallice, 1991](#)), they are generally not designed to be realistic models of biological function. The neural perspective on deep learning is motivated by two main ideas. One idea is that the brain provides a proof by example that intelligent behavior is possible, and a conceptually straightforward path to building intelligence is to reverse engineer the computational principles behind the brain and duplicate its functionality. Another perspective is that it would be deeply interesting to understand the brain and the principles that underlie human intelligence, so machine learning models that shed light on these basic scientific questions are useful apart from their ability to solve engineering applications.

The modern term “deep learning” goes beyond the neuroscientific perspective on the current breed of machine learning models. It appeals to a more general principle of learning *multiple levels of composition*, which can be applied in machine learning frameworks that are not necessarily neurally inspired.

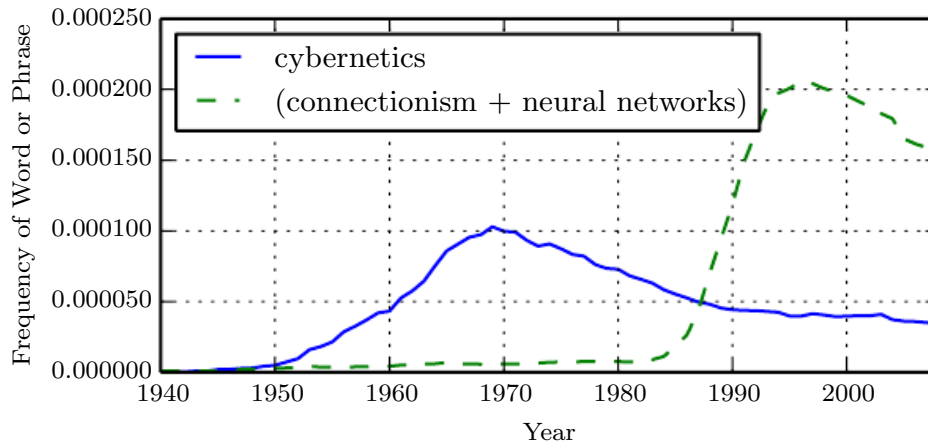


Figure 1.7: The figure shows two of the three historical waves of artificial neural nets research, as measured by the frequency of the phrases “cybernetics” and “connectionism” or “neural networks” according to Google Books (the third wave is too recent to appear). The first wave started with cybernetics in the 1940s–1960s, with the development of theories of biological learning (McCulloch and Pitts, 1943; Hebb, 1949) and implementations of the first models such as the perceptron (Rosenblatt, 1958) allowing the training of a single neuron. The second wave started with the connectionist approach of the 1980–1995 period, with back-propagation (Rumelhart *et al.*, 1986a) to train a neural network with one or two hidden layers. The current and third wave, deep learning, started around 2006 (Hinton *et al.*, 2006; Bengio *et al.*, 2007; Ranzato *et al.*, 2007a), and is just now appearing in book form as of 2016. The other two waves similarly appeared in book form much later than the corresponding scientific activity occurred.

The earliest predecessors of modern deep learning were simple linear models motivated from a neuroscientific perspective. These models were designed to take a set of  $n$  input values  $x_1, \dots, x_n$  and associate them with an output  $y$ . These models would learn a set of weights  $w_1, \dots, w_n$  and compute their output  $f(\mathbf{x}, \mathbf{w}) = x_1 w_1 + \dots + x_n w_n$ . This first wave of neural networks research was known as cybernetics, as illustrated in figure 1.7.

The McCulloch-Pitts Neuron (McCulloch and Pitts, 1943) was an early model of brain function. This linear model could recognize two different categories of inputs by testing whether  $f(\mathbf{x}, \mathbf{w})$  is positive or negative. Of course, for the model to correspond to the desired definition of the categories, the weights needed to be set correctly. These weights could be set by the human operator. In the 1950s, the perceptron (Rosenblatt, 1958, 1962) became the first model that could learn the weights defining the categories given examples of inputs from each category. The **adaptive linear element** (ADALINE), which dates from about the same time, simply returned the value of  $f(\mathbf{x})$  itself to predict a real number (Widrow and Hoff, 1960), and could also learn to predict these numbers from data.

These simple learning algorithms greatly affected the modern landscape of machine learning. The training algorithm used to adapt the weights of the ADALINE was a special case of an algorithm called **stochastic gradient descent**. Slightly modified versions of the stochastic gradient descent algorithm remain the dominant training algorithms for deep learning models today.

Models based on the  $f(\mathbf{x}, \mathbf{w})$  used by the perceptron and ADALINE are called **linear models**. These models remain some of the most widely used machine learning models, though in many cases they are *trained* in different ways than the original models were trained.

Linear models have many limitations. Most famously, they cannot learn the XOR function, where  $f([0, 1], \mathbf{w}) = 1$  and  $f([1, 0], \mathbf{w}) = 1$  but  $f([1, 1], \mathbf{w}) = 0$  and  $f([0, 0], \mathbf{w}) = 0$ . Critics who observed these flaws in linear models caused a backlash against biologically inspired learning in general (Minsky and Papert, 1969). This was the first major dip in the popularity of neural networks.

Today, neuroscience is regarded as an important source of inspiration for deep learning researchers, but it is no longer the predominant guide for the field.

The main reason for the diminished role of neuroscience in deep learning research today is that we simply do not have enough information about the brain to use it as a guide. To obtain a deep understanding of the actual algorithms used by the brain, we would need to be able to monitor the activity of (at the very least) thousands of interconnected neurons simultaneously. Because we are not able to do this, we are far from understanding even some of the most simple and