

Deep Learning

**Ian Goodfellow
Yoshua Bengio and
Aaron Courville**

**The MIT Press
Cambridge, Massachusetts
London, England**

© 2016 Massachusetts Institute of Technology

This book was set in SFRM1095 by diacriTech, Chennai.

Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Names: Goodfellow, Ian, author. | Bengio, Yoshua, author. | Courville, Aaron, author.

Title: Deep learning / Ian Goodfellow, Yoshua Bengio, and Aaron Courville.

Description: Cambridge, MA : MIT Press, [2017] | Series: Adaptive computation and machine learning series | Includes bibliographical references and index.

Identifiers: LCCN 2016022992 | ISBN 9780262035613 (hardcover : alk. paper)

Subjects: LCSH: Machine learning,

Classification: LCC Q325.5 .G66 2017 | DDC 006.3/1–dc23 LC record available at <https://lccn.loc.gov/2016022992>

Contents

Website	xiii
Notation	xix
1 Introduction	1
1.1 Who Should Read This Book?	8
1.2 Historical Trends in Deep Learning	12
I Applied Math and Machine Learning Basics	27
2 Linear Algebra	29
2.1 Scalars, Vectors, Matrices and Tensors	29
2.2 Multiplying Matrices and Vectors	32
2.3 Identity and Inverse Matrices	34
2.4 Linear Dependence and Span	35
2.5 Norms	36
2.6 Special Kinds of Matrices and Vectors	38
2.7 Eigendecomposition	39
2.8 Singular Value Decomposition	42
2.9 The Moore-Penrose Pseudoinverse	43
2.10 The Trace Operator	44
2.11 The Determinant	45

3 Probability and Information Theory	51
3.1 Why Probability?	52
3.2 Random Variables	54
3.3 Probability Distributions	54
3.4 Marginal Probability	56
3.5 Conditional Probability	57
3.6 The Chain Rule of Conditional Probabilities	57
3.7 Independence and Conditional Independence	58
3.8 Expectation, Variance and Covariance	58
3.9 Common Probability Distributions	60
3.10 Useful Properties of Common Functions	65
3.11 Bayes' Rule	68
3.12 Technical Details of Continuous Variables	68
3.13 Information Theory	70
3.14 Structured Probabilistic Models	74
4 Numerical Computation	77
4.1 Overflow and Underflow	77
4.2 Poor Conditioning	79
4.3 Gradient-Based Optimization	79
4.4 Constrained Optimization	89
4.5 Example: Linear Least Squares	92
5 Machine Learning Basics	95
5.1 Learning Algorithms	96
5.2 Capacity, Overfitting and Underfitting	107
5.3 Hyperparameters and Validation Sets	117
5.4 Estimators, Bias and Variance	119
5.5 Maximum Likelihood Estimation	128
5.6 Bayesian Statistics	132
5.7 Supervised Learning Algorithms	136
5.8 Unsupervised Learning Algorithms	142

5.9	Stochastic Gradient Descent	147
5.10	Building a Machine Learning Algorithm	149
5.11	Challenges Motivating Deep Learning	151
II	Deep Networks: Modern Practices	161
6	Deep Feedforward Networks	163
6.1	Example: Learning XOR	166
6.2	Gradient-Based Learning	171
6.3	Hidden Units	185
6.4	Architecture Design	191
6.5	Back-Propagation and Other Differentiation Algorithms	197
6.6	Historical Notes	217
7	Regularization for Deep Learning	221
7.1	Parameter Norm Penalties	223
7.2	Norm Penalties as Constrained Optimization	230
7.3	Regularization and Under-Constrained Problems	232
7.4	Dataset Augmentation	233
7.5	Noise Robustness	235
7.6	Semi-Supervised Learning	236
7.7	Multitask Learning	237
7.8	Early Stopping	239
7.9	Parameter Tying and Parameter Sharing	246
7.10	Sparse Representations	247
7.11	Bagging and Other Ensemble Methods	249
7.12	Dropout	251
7.13	Adversarial Training	261
7.14	Tangent Distance, Tangent Prop and Manifold Tangent Classifier .	263
8	Optimization for Training Deep Models	267
8.1	How Learning Differs from Pure Optimization	268

8.2	Challenges in Neural Network Optimization	275
8.3	Basic Algorithms	286
8.4	Parameter Initialization Strategies	292
8.5	Algorithms with Adaptive Learning Rates	298
8.6	Approximate Second-Order Methods	302
8.7	Optimization Strategies and Meta-Algorithms	309
9	Convolutional Networks	321
9.1	The Convolution Operation	322
9.2	Motivation	324
9.3	Pooling	330
9.4	Convolution and Pooling as an Infinitely Strong Prior	334
9.5	Variants of the Basic Convolution Function	337
9.6	Structured Outputs	347
9.7	Data Types	348
9.8	Efficient Convolution Algorithms	350
9.9	Random or Unsupervised Features	351
9.10	The Neuroscientific Basis for Convolutional Networks	353
9.11	Convolutional Networks and the History of Deep Learning	359
10	Sequence Modeling: Recurrent and Recursive Nets	363
10.1	Unfolding Computational Graphs	365
10.2	Recurrent Neural Networks	368
10.3	Bidirectional RNNs	383
10.4	Encoder-Decoder Sequence-to-Sequence Architectures	385
10.5	Deep Recurrent Networks	387
10.6	Recursive Neural Networks	388
10.7	The Challenge of Long-Term Dependencies	390
10.8	Echo State Networks	392
10.9	Leaky Units and Other Strategies for Multiple Time Scales	395
10.10	The Long Short-Term Memory and Other Gated RNNs	397

10.11 Optimization for Long-Term Dependencies	401
10.12 Explicit Memory	405
11 Practical Methodology	409
11.1 Performance Metrics	410
11.2 Default Baseline Models	413
11.3 Determining Whether to Gather More Data	414
11.4 Selecting Hyperparameters	415
11.5 Debugging Strategies	424
11.6 Example: Multi-Digit Number Recognition	428
12 Applications	431
12.1 Large-Scale Deep Learning	431
12.2 Computer Vision	440
12.3 Speech Recognition	446
12.4 Natural Language Processing	448
12.5 Other Applications	465
III Deep Learning Research	475
13 Linear Factor Models	479
13.1 Probabilistic PCA and Factor Analysis	480
13.2 Independent Component Analysis (ICA)	481
13.3 Slow Feature Analysis	484
13.4 Sparse Coding	486
13.5 Manifold Interpretation of PCA	489
14 Autoencoders	493
14.1 Undercomplete Autoencoders	494
14.2 Regularized Autoencoders	495
14.3 Representational Power, Layer Size and Depth	499
14.4 Stochastic Encoders and Decoders	500

14.5	Denoising Autoencoders	501
14.6	Learning Manifolds with Autoencoders	506
14.7	Contractive Autoencoders	510
14.8	Predictive Sparse Decomposition	514
14.9	Applications of Autoencoders	515
15	Representation Learning	517
15.1	Greedy Layer-Wise Unsupervised Pretraining	519
15.2	Transfer Learning and Domain Adaptation	526
15.3	Semi-Supervised Disentangling of Causal Factors	532
15.4	Distributed Representation	536
15.5	Exponential Gains from Depth	543
15.6	Providing Clues to Discover Underlying Causes	544
16	Structured Probabilistic Models for Deep Learning	549
16.1	The Challenge of Unstructured Modeling	550
16.2	Using Graphs to Describe Model Structure	554
16.3	Sampling from Graphical Models	570
16.4	Advantages of Structured Modeling	572
16.5	Learning about Dependencies	572
16.6	Inference and Approximate Inference	573
16.7	The Deep Learning Approach to Structured Probabilistic Models .	575
17	Monte Carlo Methods	581
17.1	Sampling and Monte Carlo Methods	581
17.2	Importance Sampling	583
17.3	Markov Chain Monte Carlo Methods	586
17.4	Gibbs Sampling	590
17.5	The Challenge of Mixing between Separated Modes	591
18	Confronting the Partition Function	597
18.1	The Log-Likelihood Gradient	598
18.2	Stochastic Maximum Likelihood and Contrastive Divergence . . .	599

18.3	Pseudolikelihood	607
18.4	Score Matching and Ratio Matching	609
18.5	Denoising Score Matching	611
18.6	Noise-Contrastive Estimation	612
18.7	Estimating the Partition Function	614
19	Approximate Inference	623
19.1	Inference as Optimization	624
19.2	Expectation Maximization	626
19.3	MAP Inference and Sparse Coding	627
19.4	Variational Inference and Learning	629
19.5	Learned Approximate Inference	642
20	Deep Generative Models	645
20.1	Boltzmann Machines	645
20.2	Restricted Boltzmann Machines	647
20.3	Deep Belief Networks	651
20.4	Deep Boltzmann Machines	654
20.5	Boltzmann Machines for Real-Valued Data	667
20.6	Convolutional Boltzmann Machines	673
20.7	Boltzmann Machines for Structured or Sequential Outputs	675
20.8	Other Boltzmann Machines	677
20.9	Back-Propagation through Random Operations	678
20.10	Directed Generative Nets	682
20.11	Drawing Samples from Autoencoders	701
20.12	Generative Stochastic Networks	704
20.13	Other Generation Schemes	706
20.14	Evaluating Generative Models	707
20.15	Conclusion	710
	Bibliography	711
	Index	767

Website

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.

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
$e^{(i)}$	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}
\mathbf{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
$\mathbb{E}_{x \sim P}[f(x)]$ or $\mathbb{E}f(x)$	Expectation of $f(x)$ with respect to $P(x)$
$\text{Var}(f(x))$	Variance of $f(x)$ under $P(x)$
$\text{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(x)$
$H(x)$	Shannon entropy of the random variable x
$D_{\text{KL}}(P \ Q)$	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_{data}	The data generating distribution
\hat{p}_{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 $\mathbf{X}_{i,:}$

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 through its relation to simpler concepts. By gathering knowledge from experience, this approach avoids the need for human operators to formally specify all the knowledge that the computer needs. The hierarchy of concepts enables 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 only recently have begun 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 automatically about statements in these formal languages 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 enabled 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 how features are defined in any way. If logistic regression were 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 the speaker's vocal tract. This feature gives a strong clue as to whether the speaker is a man, woman, or child.

For many tasks, however, 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.

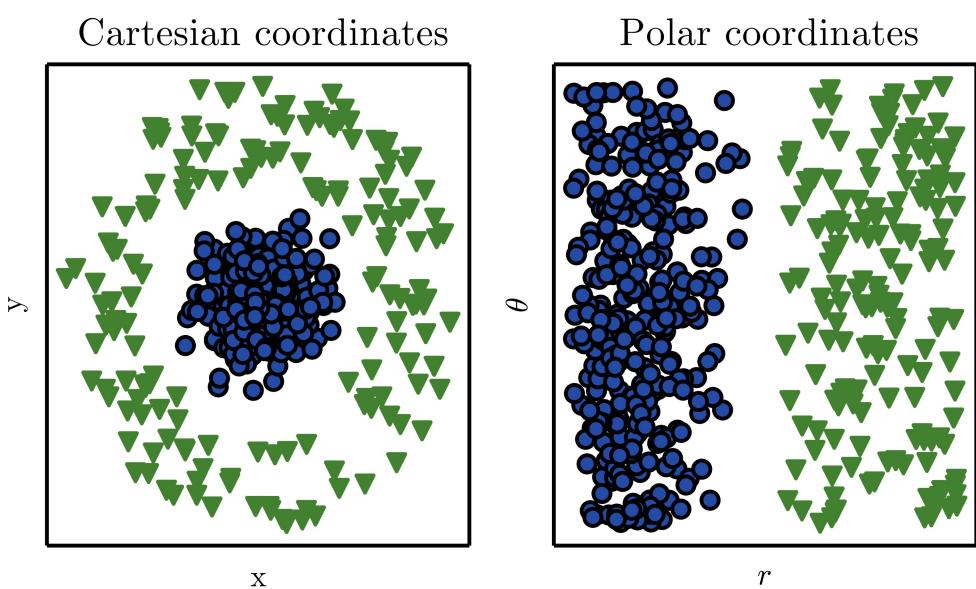


Figure 1.1: Example of different representations: suppose we want to separate two categories of data by drawing a line between them in a scatterplot. In the plot on the left, we represent some data using Cartesian coordinates, and the task is impossible. In the plot on the right, we represent the data 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 enable 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 for 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, which converts the input data into a different representation, and a **decoder** function, which 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 they 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 as either 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 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 enables 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 enables the computer to learn a multistep 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 instructions can refer back to the results of earlier instructions.

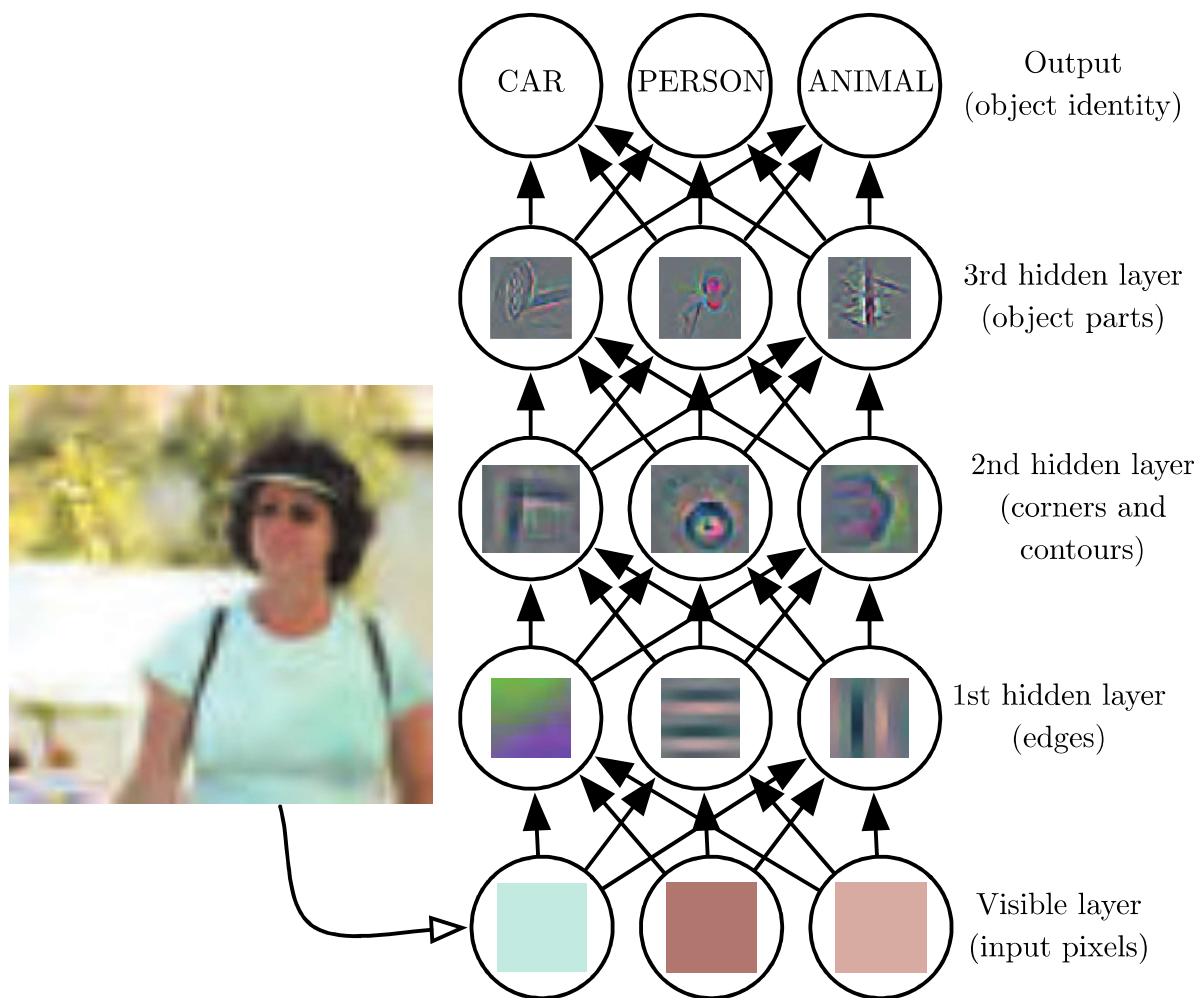


Figure 1.2: Illustration of a deep learning model. It is difficult for a computer to understand the meaning of raw sensory input data, such as this image represented as a collection of pixel values. The function mapping from a set of pixels to an object identity is very complicated. Learning or evaluating 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).

According to this view of deep learning, not all 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

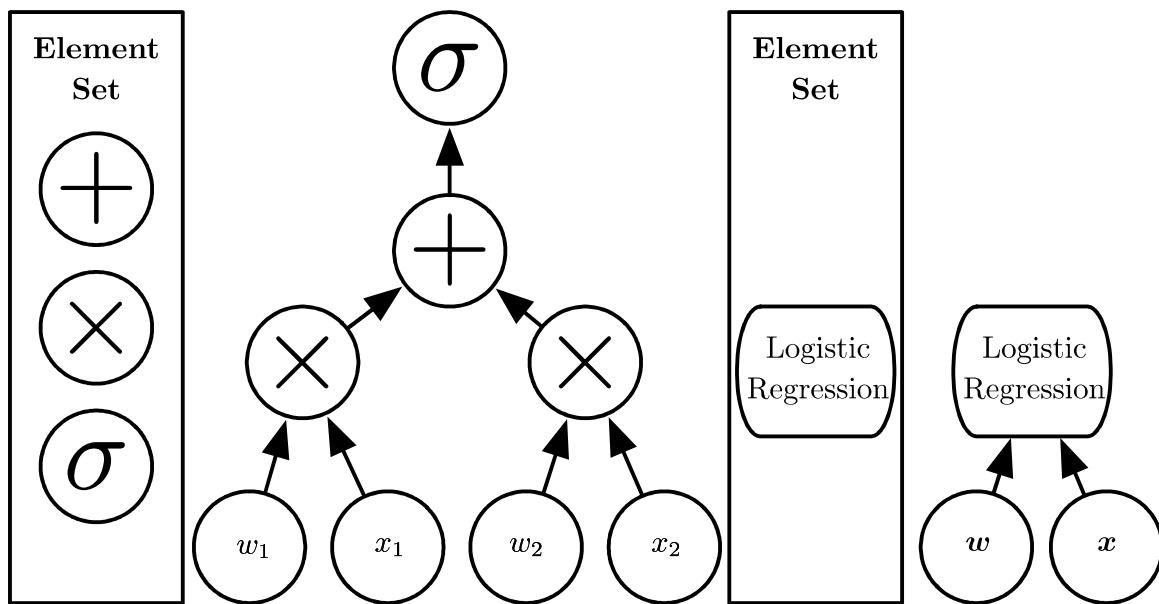


Figure 1.3: Illustration of computational graphs mapping an input to an output where each node performs an operation. Depth is the length of the longest path from input to output but depends on the definition of what constitutes a possible computational step. The computation depicted in these graphs is the output of a logistic regression model, $\sigma(\mathbf{w}^T \mathbf{x})$, where σ 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.

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 see only one eye. After detecting that a face is present, the system can then infer that a second eye is probably present as well. In this case, the graph of concepts includes only 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 be safely regarded as the study of models that involve a greater amount of composition of either 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 enables computer systems to improve with experience and data. We contend that 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 representing 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 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 proved useful in many soft-

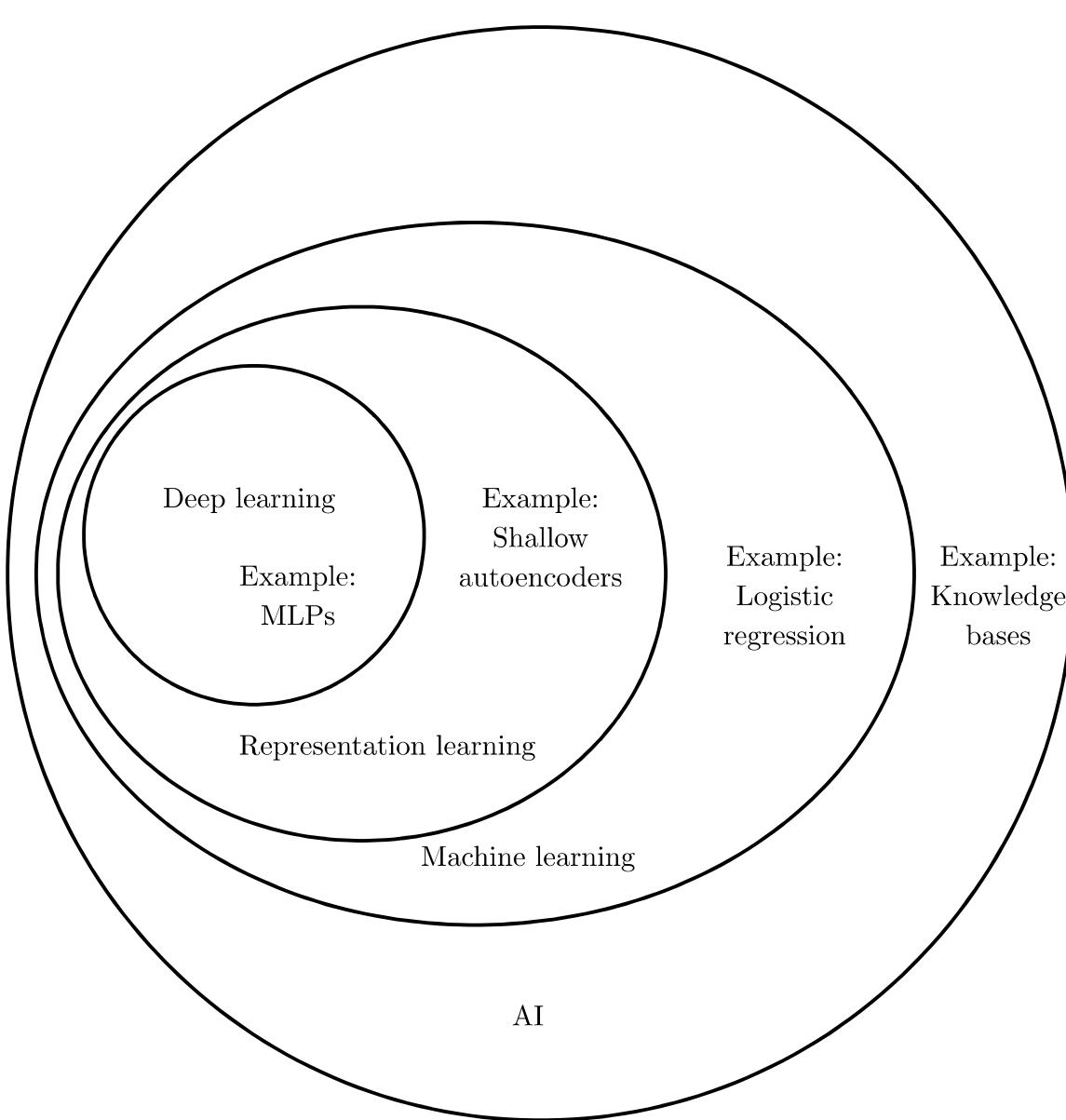


Figure 1.4: A Venn diagram showing how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is used for many but not all approaches to AI. Each section of the Venn diagram includes an example of an AI technology.

ware 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 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, which 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 those who just want

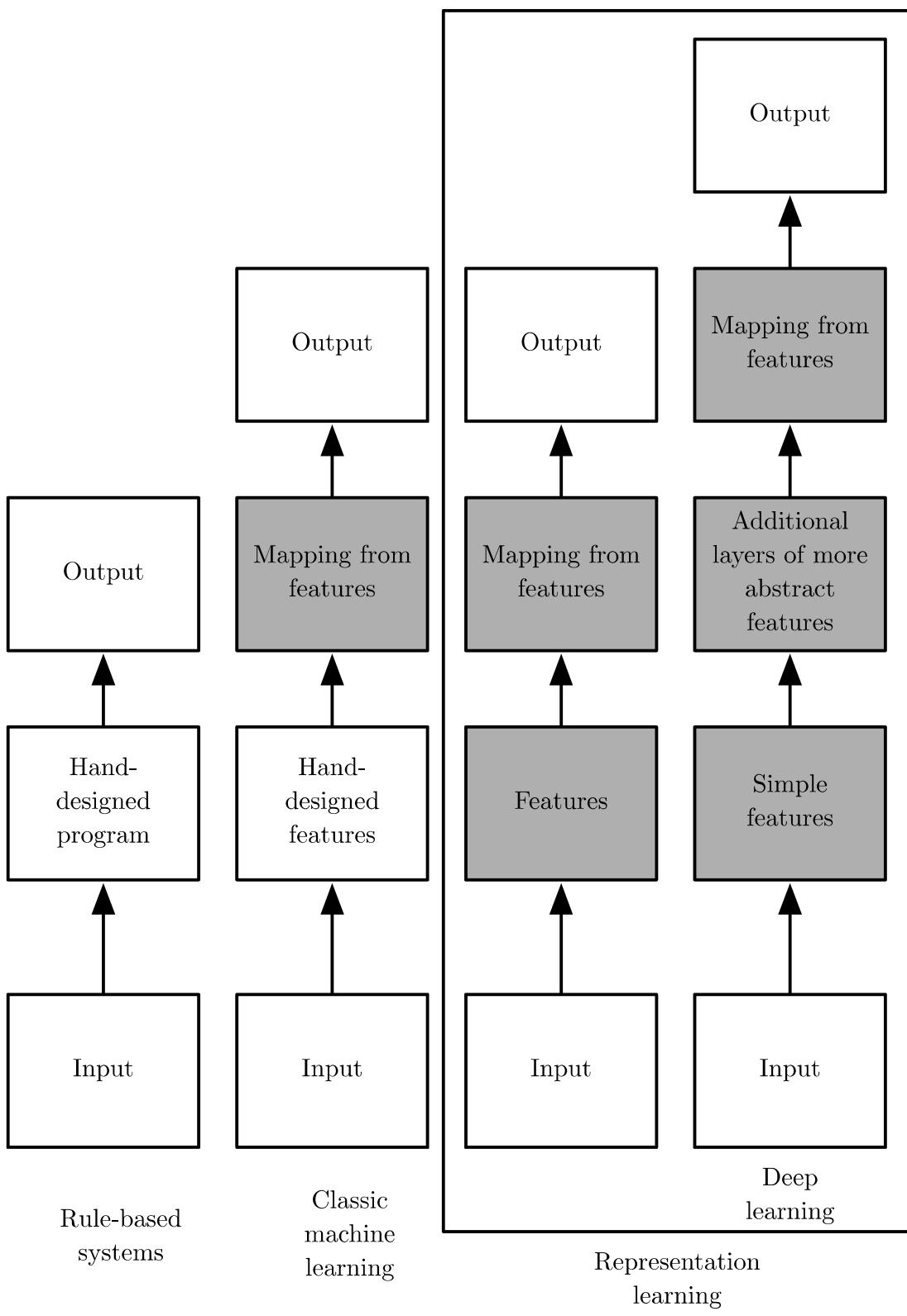


Figure 1.5: Flowcharts showing how the different parts of an AI system relate to each other within different AI disciplines. Shaded boxes indicate components that are able to learn from data.

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.

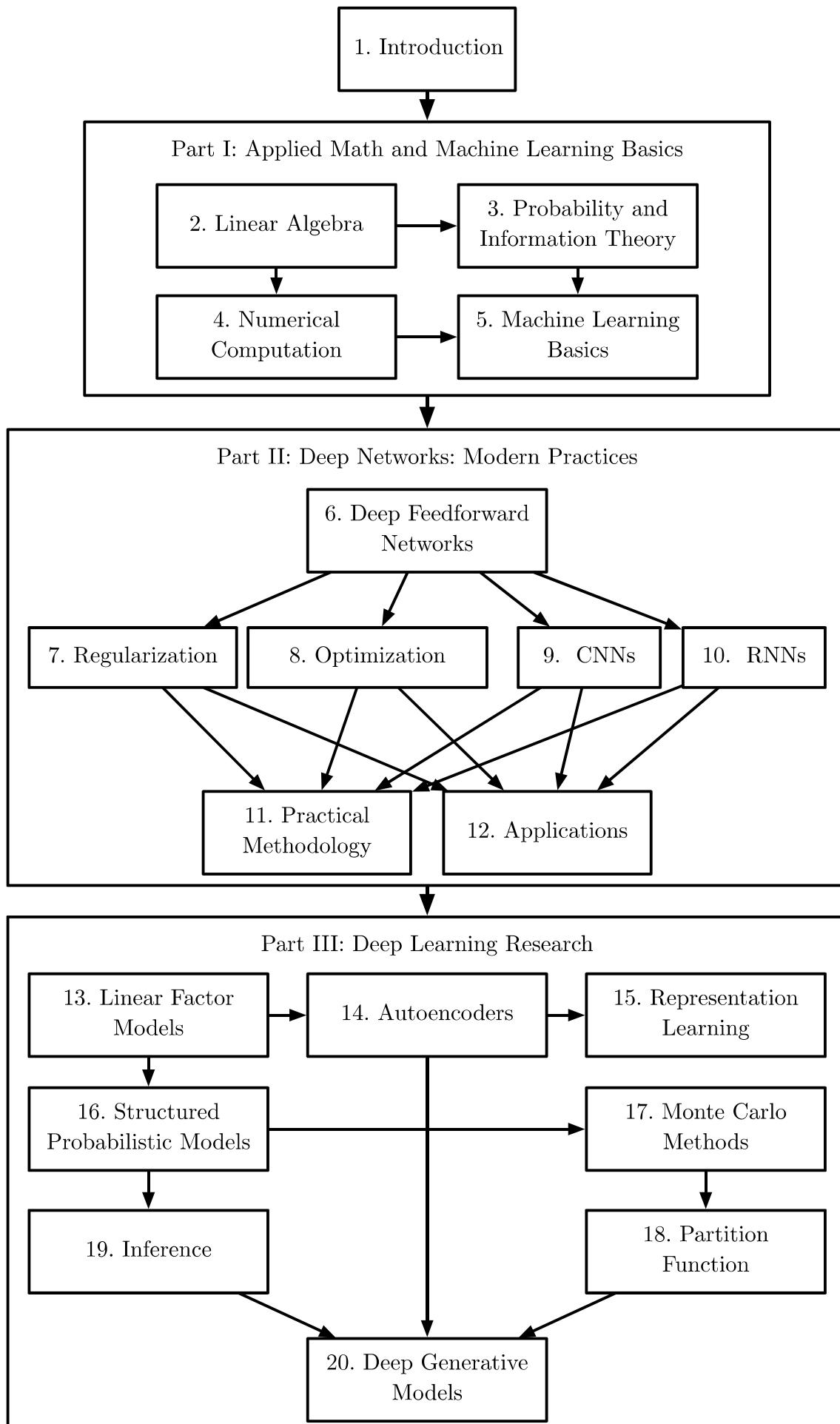


Figure 1.6: The high-level organization of the book. An arrow from one chapter to another indicates that the former chapter is prerequisite material for understanding the latter.

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.

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, only recently being 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. Some basic context, however, is useful for understanding deep learning. Broadly speaking, there have been three waves of development: 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.