1

Introduction to Deep Learning

LEARNING OBJECTIVES

- Be familiar with types of machine learning.
- To understand data representation in machine learning.
- To understand basic concepts on dataset: data examples, features, and label.
- To understand three types of dataset: training set, testing set, and validation set.
- Be aware of deep learning history. Overview of deep learning.
- To know resources for learning deep learning.

1.1 Introduction

Over the past years, artificial intelligence (AI) has walked into every area of our lives. If you are a writer and want to write a poem in praise of AI in the style of Shakespeare, you just need to type: *write a poem in praise of AI in the style of Shakespeare,* and OpenAI ChatGPT (released in 2022) (https://chat.openai.com/chat) will return a poem to you like this:

> *Oh AI, thou art a wonder to behold*
> *Thy algorithms and codes, so finely tuned and bold*
> *Thou canst solve problems and answer questions with ease*
> *Thy boundless knowledge, a vast and infinite library to release*
> ...

If you are a programmer, ChatGPT can write code for you. If you want a paragraph to be translated into a different language, you can ask ChatGPT to do it. If you are a Go player and want to improve your game, you can ask AlphaGo (a different AI program) (https://www.deepmind.com/research/highlighted-research/alphago) for the best move.

ChatGPT uses deep learning algorithms to generate text responses to prompts. It is based on a transformer model that uses self-attention mechanisms to process and generate text. The ChatGPT model has been trained on a massive corpus of text data, which includes a wide range of topics and styles. As a result, the model is able to generate responses that are highly relevant to the prompt and exhibit a level of knowledge and understanding similar to a human.

Deep Learning: Principles and Implementations, First Edition. Weidong "Will" Kuang.

AlphaGo is a computer program that consistently defeated Go world champions in different global arenas and has become the strongest Go player in history. Its model combines an advanced search tree with deep neural networks and is trained through reinforcement learning (RL).

Deep learning was first introduced to machine learning in the 1980s. The rapid development of deep learning in recent years is a result of the significant advances in computer technologies and the availability of a large amount of data. The graphics processing units (GPUs), designed originally for graph visualization, provide powerful computational ability on matrix data, which is a predominant data structure in deep learning. The combination of powerful computation and large memory makes it possible to train large neural networks with billions of parameters. In the past, without sufficient training data, there was no way to find *good* parameters for a large neural network. Over the past two decades, massive amounts of data (e.g. images, videos, and social network data) have been generated through the internet and mobile electronic devices.

The goal of this book is to present the fundamentals of neural networks and deep learning. In this chapter, we begin with necessary terminologies and general formulation of machine learning. Then we present a brief overview of deep learning and how key ingredients of deep learning are linked to the chapters in this book. Finally, we provide some useful resources for deep learning.

1.2 Types of Machine Learning

The relationship of three terms, *artificial intelligence*, *machine learning*, and *deep learning*, can be illustrated by Figure 1.1, where AI refers to the technologies of mimicking the intelligence or behavioral pattern of humans or any other living entity, machine learning refers to a computer program by which a computer can learn from data without using a complex set of rules, and deep learning is a subset of machine learning that is inspired by the neural networks in human's brain.

Although the phrase "machine learning" has become more and more frequently used, it is challenging to give a perfect definition for it. In the 1950s, Arthur Samuel defined machine learning as the field of study that gives computers the ability to learn without being explicitly programmed. A recent widely accepted definition of machine learning was given by Tom Mitchell (Mitchell 1997), as

> *A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.*

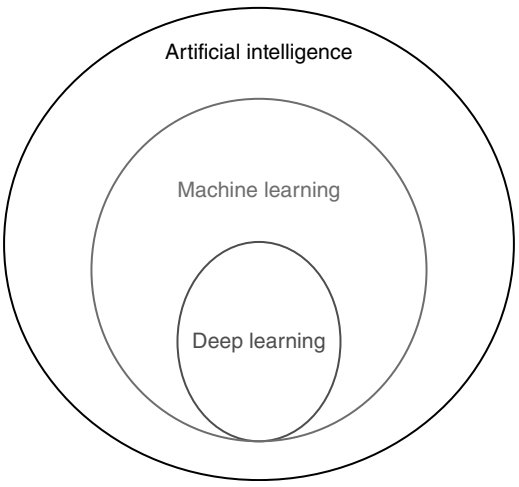


Figure 1.1 Artificial intelligence, machine learning, and deep learning.

For example, an email spam filter is a computer program that predicts an incoming email as spam or not spam, based on many samples of spam emails and non-spam emails. In this setting, the task T is to classify emails as spam or not spam, the experience E refers to learning from emails already identified by humans as spam or not spam, and the performance measure P can be the percentage of emails correctly predicted by the program as spam or not spam.

The general characteristics of machine learning can be summarized as follows:

- *Learning from data*. The data can be viewed as experiences that have already happened.
- *Adapting*. The machine takes an action on the data and gets an output. If the output is not *good* enough, the machine should modify itself so that the output is better.
- *Generalizing*. The knowledge learned by the machine should be applied to future situations. By generalizing, the machine can *understand* new data that has never been seen before.

When we discuss or develop a specific machine learning system, it is helpful or necessary to identify the type of system. Machine learning systems can be classified according to the amount and type of supervision available during training, roughly into three basic types: *supervised learning*, *unsupervised learning*, and *reinforcement learning*.

1.2.1 Supervised Learning

Supervised learning is the most common type of machine learning. Literally, it means that the learning process is supervised by a labeled dataset. Specifically, we train a *model* by feeding it with a *dataset*. The training dataset is a set of sample points with each sample point consisting of *features* and *label*. For each sample point, its features represent its properties and are used as the input of the model while the label represents the desired output of the model. The model usually has many parameters with randomly initialized values. The goal of the learning process (also called training process) is to search for the best values of the parameters so that the outputs of the model are *close* to the given labels for the inputs. Furthermore, a successful model should have a capability of *generalization*, i.e. the trained model should produce sensible output for any input that was not encountered in the training dataset. This process is called inference. Thus, the development of the learning system involves two processes: *training* and *inference*, as illustrated in Figure 1.2.

In the supervised learning setting, the training dataset is a set of samples, denoted as $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}), i = 1, 2, ..., m\}$. The superscript (i) indicates the data point as the ith sample, $i = 1, 2, ... m$, where m is the total number of data samples, called the size of the training dataset. $\mathbf{y}^{(i)}$ is the label (also known as target or ground truth) corresponding to the input feature $\mathbf{x}^{(i)}$. By learning from the m labeled data samples, the machine learning algorithm will find the approximate relationship between the inputs and the labels, so that it can predict the label for a new input $\mathbf{x}$, assuming that $\mathbf{x}$ is drawn from the underlying distribution of the training dataset. In general, $\mathbf{x}^{(i)}$ is a vector or high-

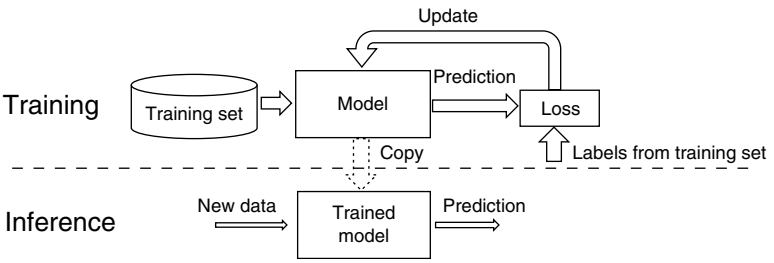
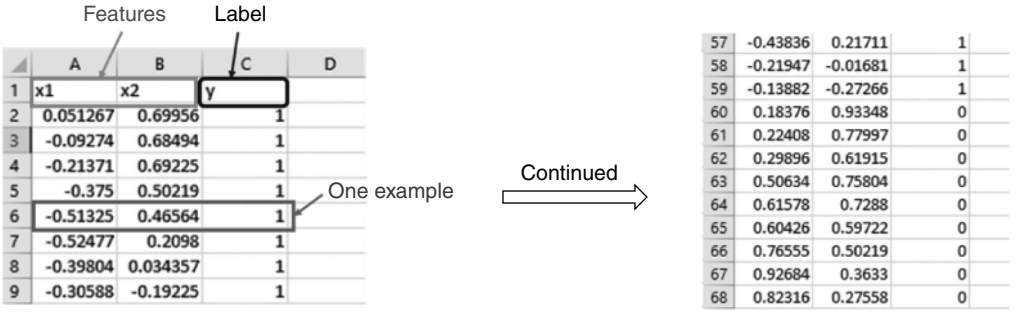


Figure 1.2 Training versus inference in supervised learning.



	A	B	C	D
1	x1	x2	y	
2	0.051267	0.69956	1	
3	-0.09274	0.68494	1	
4	-0.21371	0.69225	1	
5	-0.375	0.50219	1	
6	-0.51325	0.46564	1	
7	-0.52477	0.2098	1	
8	-0.39804	0.034357	1	
9	-0.30588	-0.19225	1	

57	-0.43836	0.21711	1
58	-0.21947	-0.01681	1
59	-0.13882	-0.27266	1
60	0.18376	0.93348	0
61	0.22408	0.77997	0
62	0.29896	0.61915	0
63	0.50634	0.75804	0
64	0.61578	0.7288	0
65	0.60426	0.59722	0
66	0.76555	0.50219	0
67	0.92684	0.3633	0
68	0.82316	0.27558	0

Figure 1.3 A dataset with two features and a scalar label.

dimensional matrix while $\mathbf{y}^{(i)}$ is a scalar or vector. Figure 1.3 illustrates a simple dataset with the input vector $x^{(i)} \in \mathbb{R}^2$ (i.e. two features) and the output label $y \in \mathbb{R}$. In practice the number of input features is typically much larger. For example, in computer vision, a 224×224 RGB (Red-Green-Blue) color image has $224 \times 224 \times 3 = 150{,}528$ features.

Supervised learning can be further categorized into regression and classification. In a regression task, the labels are continuously valued. In other words, the model maps input variables to some continuous function. In a classification task where the labels are discrete, the model predicts the discrete label associated with the category to which the input sample belongs.

An example of regression is the prediction of a house price, given a training dataset containing information about many houses (e.g. area, number of bedrooms, age, and number of floors) and their market prices. For a house, its properties make up the input features while its market price is its label. A trained model is expected to predict (or estimate) the price for a house based on its properties. For visualization, we consider only one feature of the house – area. The sample points (area, price) can be plotted as solid circles in Figure 1.4(a). To predict the price for a house with a certain area, the goal of the regression task is to find a curve (dashed curve) to fit the data samples according to a criterion, e.g. minimal mean squared error (MSE). Based on the fitting curve, we can predict the price for a house with any area A, shown by the dashed circle. The model can be

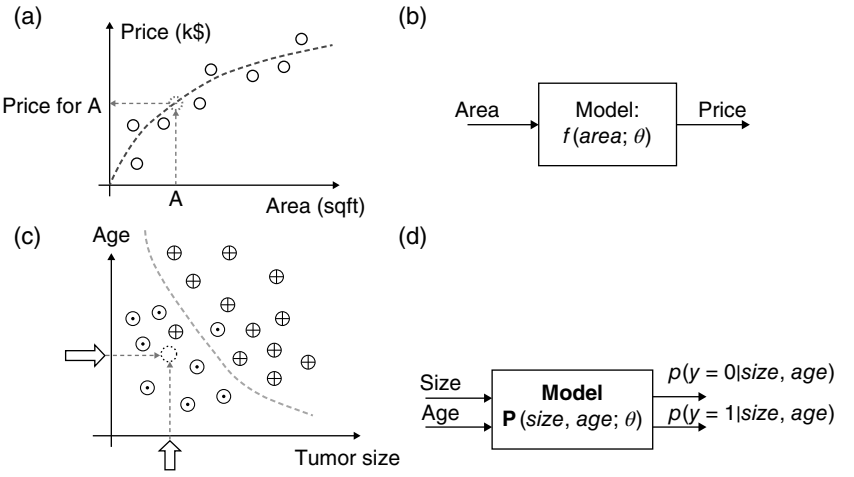


Figure 1.4 Regression versus classification: (a) regression, (b) model for regression, (c) classification, and (d) model for classification.

mathematically described by a function $f(area; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the model parameter. The output of the function is the house price, shown in Figure 1.4(b).

An example of classification is illustrated in Figure 1.4(c), which is the classification of breast cancer, given a training dataset including many patients' features and labels. The labels indicate the types of breast cancer (label $y = 1$ for malignant, $y = 0$ for benign). A machine learning model will learn from this training set, and then be able to predict the type of breast cancer ($y = 1$ or $y = 0$?) for a new patient. To visualize the concept, we consider two features: age and tumor size. The training data samples are plotted as $\odot$ for benign samples and $\oplus$ for malignant samples. After learning from the training dataset, the model will be able to predict the category for any pair (tumor size, age). Thus, a decision boundary (indicated by the dashed curve), which separates two types of breast cancer, exists for a trained model. In the case of Figure 1.4(c), the new data sample indicated by the dashed circle is classified as benign by the model because it is located below downside of the decision boundary. In practice, to predict the discrete label, the model estimates the conditional probability of each category conditioned on the input, e.g. $p(y = 0 \mid size, age)$ that represents the probability of $y = 0$ given a pair of values (size, age). The category with the largest probability across the outputs is predicted, as shown in Figure 1.4(d).

The applications of supervised learning include, but are not limited to, image classification, object detection (in computer vision), facial recognition, tagging, recommender systems, speech recognition, text-to-speech, question answering, and machine translation.

1.2.2 Unsupervised Learning

Suppose that the training data is not labeled, then a machine learning algorithm is expected to learn the structural pattern of the data. This type of learning is called unsupervised learning. For instance, the structure can be obtained by clustering data samples based on relationships among the input features. The model tries to quantify the similarity between data samples so that the samples that have something in common or close are grouped together. Figure 1.5 shows the unlabeled data samples. Based on the distribution of samples, a possible clustering result is that they are divided into two groups indicated by two dashed circles.

For instance, the following tasks belong to unsupervised learning:

a) *Clustering*. Take a collection of 1,000,000 different genes and find a way to automatically group these genes in terms of the similarity in features, such as lifespan, location, roles, and so on.
b) *Computer clusters*. Consider a large data center with a large number of computers. We try to organize the computers into different clusters (or groups) based on their roles, so that the data center works efficiently.
c) *Social network analysis*. Given the activities of a person in social networks such as emails, Facebook, Instagram, and WeChat, we try to analyze the nature of their social groups.
d) *Market segmentation*. Given a huge database of customer information, we try to group customers into different market segments so that we can sell products more efficiently.

Visualization and dimensionality reduction are also classified as unsupervised learning. These algorithms try to transform higher dimensional data into lower dimensional (even 2D or 3D) representation with a minimum information loss. Reducing data dimension can remove redundancy among features of input, and thus improve the efficiency of the subsequent machine learning algorithm.

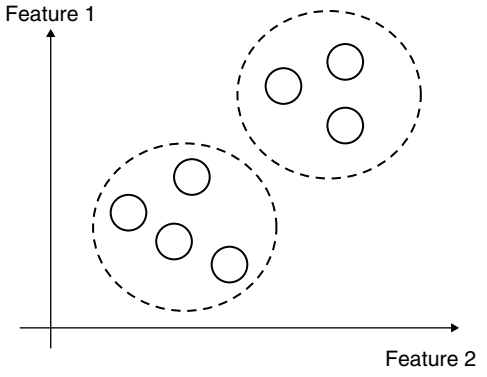


Figure 1.5 Unsupervised learning: clustering.

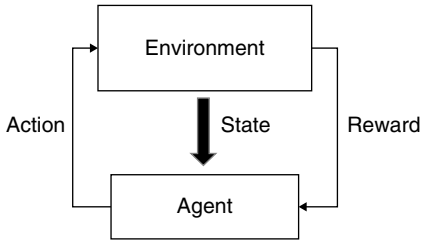


Figure 1.6 Reinforcement learning.

1.2.3 Reinforcement Learning

In reinforcement learning, we want to develop an *agent* (or an algorithm) that can intelligently interact with an *environment*, as illustrated in Figure 1.6. At each time-step, the agent selects an *action* from a set of actions and applies it to the environment. Consequently, the environment changes from one *state* to another state, which can be observed by the agent, and returns a *reward* to the agent. During the training process, the agent learns from a series of interactions with the environment via an exploratory trial-and-error approach. The goal is to find the best strategy, called a *policy*, to get the most reward over time on average. A policy defines what action the agent should take for a given state.

1.3 Data Representation in Machine Learning

This section presents a typical way about how data are organized for machine learning.

1.3.1 Tensor

In machine learning, data are usually organized in a data structure, called *tensor*, which is a generalization of vectors and matrices and can be easily understood as a multidimensional array. A vector is a one-dimensional (1D) tensor, and a matrix is a two-dimensional (2D) tensor. Each dimension in a tensor is associated with one *axis*. The data type of the element in a tensor can be integer, floating-point, or complex numbers.

For example, [0.12, 0.34, 0.04, 1.86] is a vector with a size of 4. [[1,2], [3,4], [5,6], [7,8]] is a 2D tensor with the first axis (axis 0) size 4 and the second axis (axis 1) size 2, and thus its shape is denoted as size (4,2). The *shape* of a tensor is a tuple of integers that describes how many elements of the tensor along each axis.

A set of m samples can be represented by a single tensor. For instance, m vectors can be represented by a tensor with a shape of (m, n), where n is the length of a vector, i.e. the number of features. Time series data or sequence data can be represented by a 3D tensor of shape (m, timesteps,

features), where m is the number of sequences. One image is usually represented as a 3D tensor with a shape of (C, H, W), where C is the number of channels (e.g. for RGB image $C = 3$, each element in the tensor represents a value for one color at one pixel, illustrated in Figure 1.7), H is the height, and W is the width. A video is considered as a set of F frames, which is represented by a 4D tensor with a shape of (F, C, H, W). The batch of m videos can be represented by a 5D tensor with a shape of (m, F, C, H, W), where F is the number of frames in each video.

Of course, the order of dimensions can be arbitrary. For example, the shape of an image can be (H, W, C).

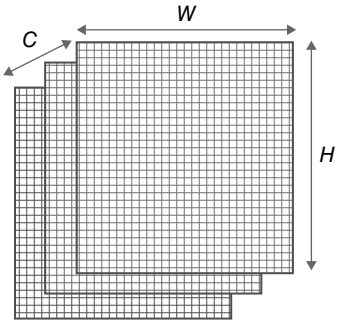


Figure 1.7 Data structure of an image with three colors (red, green, blue).

1.3.2 Datasets: Training, Validation, and Testing

Data samples are usually stored in one or multiple files. To develop a machine learning model, we partition the entire set of data samples into three datasets: **training set**, **validation set**, and **testing set**. Ideally, all data samples are generated under the same statistical condition and have the same format. In other words, all three datasets have the same statistical properties.

In general, the process of developing a machine learning product consists of three steps, as shown in Figure 1.8:

Training. After studying the problem, we select different machine learning models (including different types, different hyperparameters e.g. the number of layers for neural network) and train each model using the same training set.

Validation. We test each model using the validation set and select the best model as the final model.

Testing. The final model will be tested by the testing set so that we can estimate how well the model works after it is deployed.

In many situations, obtaining a large amount of data (especially labeled data) is expensive. This leads to limited amount of available data. Thus, it is essential to use the available data efficiently. The question is: given an entire dataset, how do we split it into three sets: training, validation, testing? A basic guideline to create three datasets is illustrated by Figure 1.9, and described below.

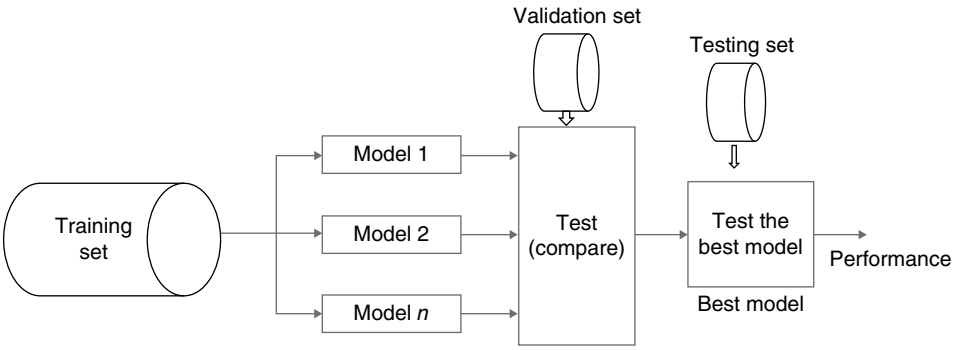


Figure 1.8 Development of a machine learning product.

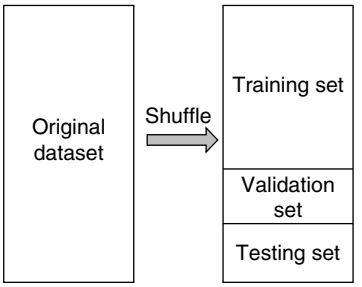


Figure 1.9 Generate training set, validation set, and testing set from the original dataset.

Ratio. Generally, the exact size ratio of three sets depends on the total amount of data and a specific case. Here the size of a set is defined as the number of data samples. Empirically, the percentages of three parts are typically 60%, 20%, and 20% for training, validation, and testing sets, respectively. However, if the number of data samples is big (e.g. more than 1,000,000), the percentages for validation and testing sets are expected to be reduced so that more data can be used for training. For example, for a dataset with a size of 1,000,000, a reasonable partition would be 980,000 (98% training), 10,000 (1% validation), and 10,000 (1% testing).

Randomness. The entire dataset is usually randomly divided into three parts: training, validation, and testing sets. This can be done by randomly reordering the data first or by assigning each data sample randomly to one of the three sets. Furthermore, it is a common practice to maintain an approximate same categorical ratio in each set, which is called stratified sampling. For example, in a classification task of breast cancer, in the entire dataset, there are 10% samples belonging to the malignant class and 90% samples in the benign class. The training set, validation set, and testing set all expect to have 10% malignant samples and 90% benign samples. Stratified sampling is important especially for the dataset with imbalanced classes.

1.3.3 Resources of Datasets

The UC Irvine Machine Learning Repository (https://archive.ics.uci.edu/ml/index.php) is a collection of databases, domain theories, and data generators that are used by the machine learning community for the empirical analysis of machine learning algorithms. It has been widely used by students, educators, and researchers all over the world as a primary source of machine learning datasets. To date, the repository has maintained more than 600 datasets as a service to the community. The popular datasets used in many machine learning books, such as Iris, Wine Quality, and Breast Cancer Wisconsin, are available at the repository website.

Kaggle (https://www.kaggle.com/) is an online community of data scientists and machine learning practitioners. Kaggle allows users to share or download datasets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges. Many classical datasets can be found on Kaggle.

1.4 An Overview of Deep Learning

In this section, we present an overview of deep learning and highlight the fundamental deep learning topics covered in this book.

Deep learning is a subfield of machine learning that is inspired by the structure and function of the brain, specifically the neural networks. Deep learning models are made up of layers of artificial neurons that are connected to each other, allowing them to learn and make decisions based on the input data. These models are trained using large amounts of data and can learn to identify patterns and relationships that are not easily recognizable by traditional methods.

1.4.1 Perceptron

The history of deep learning can be traced back to the 1940s and 1950s, when researchers first began exploring the idea of building AI based on the structure and function of the brain. The earliest form of neural network was the perceptron (Rosenblatt 1958), developed by Frank Rosenblatt in the late 1950s. The perceptron was a simple model consisting of a single layer of artificial neurons and was used for binary classification tasks. The mathematical model of a neuron in the perceptron is illustrated in Figure 1.10, and defined by

$$z = \sum_{i=1}^{n} w_i x_i \tag{1.1a}$$

$$h = g(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{if } z < \theta \end{cases} \tag{1.1b}$$

where x_i, $i = 1, 2 \ldots n$, are n input features and w_i is the corresponding weight for x_i. The neuron sums the weighted inputs. If this sum is greater than a predefined threshold θ, then the neuron fires (i.e. $h = 1$); otherwise, it does not fire (i.e. $h = 0$). This threshold function is called an *activation* function.

A perceptron can be used to implement a logic function gate, as shown in Fig. 1.11, where the circle represents a neuron that includes both the adder and the activation function defined in Eq. (1.1a) and (1.1b), and b is a bias that can be viewed as a special input with a fixed weight (-1). Thus, the perceptron has three parameters to learn: w_1, w_2, b. The process of learning (or training) is to find a set of values for these parameters so that the input-output relationship matches the truth table. For example, the perceptron works as an OR gate if $w_1 = 2$, $w_2 = 2$, and $b = 1$.

Although the perceptron with a single neuron has very limited applications in practice, it inspired the development of neurons in modern deep learning architectures, and is a good starting point for us to understand the basic concepts of neural networks. **Chapter 2** (Linear Regression) treats linear

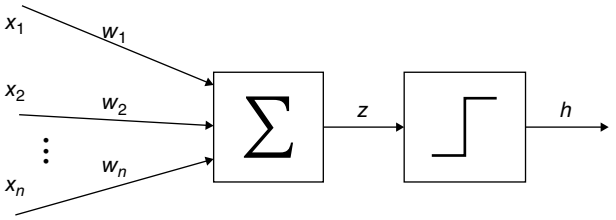
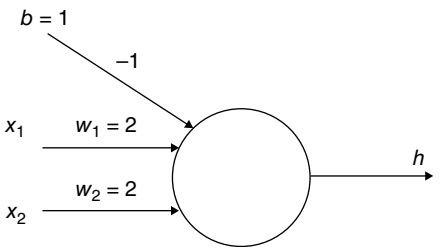


Figure 1.10 A neuron in perceptron.



x_1	x_2	h
0	0	0
0	1	1
1	0	1
1	1	1

Figure 1.11 The perceptron for OR gate.

regression corresponding to the adder in the neuron while **Chapter 3** (Logistic Regression) is dedicated to logistic regression, which is a neuron with a smooth activation function, such as the sigmoid function (see Eq.(1.3)).

1.4.2 Multilayer Neural Networks and Backpropagation

In the 1960s and 1970s, researchers continued to explore the potential of neural networks, but progress was hindered by a lack of computational power and the limitations of the models. Despite these limitations, some researchers developed more complex models, such as multilayer networks, which could solve more advanced problems. It was not until the late 1980s and early 1990s that advances in computational power and the development of new algorithms made it possible to train deep neural networks, consisting of multiple layers of artificial neurons. These deep networks were able to learn and represent more complex patterns and relationships in the data, and they began to achieve breakthrough performance in a wide range of applications, such as image and speech recognition.

An instance of a feedforward multilayer network is illustrated in Figure 1.12. The network has an input layer that just stores the input data with no operation, two hidden layers, and an output layer. All nodes, except those in the input layer, are neurons. Each layer typically consists of multiple identical neurons. Each neuron receives signals from nodes in the previous layer. In other words, the output of each neuron in one layer is connected to every neuron in the next layer. In general, we say that these layers are fully connected (FC). An adjustable parameter, called *weight*, is associated with each connection. However, some practical neural networks (e.g. convolutional neural networks [CNNs]) are sparsely connected, and can be understood as special cases where most connections have a weight of zero.

The output of the output layer $(y_1, y_2, ..., y_K)$ is the prediction of the label (or ground truth) for the current input.

To train the neural network, i.e. to find *good* values for the weights, we compute an objective function that measures the error (or distance) between the output prediction and the desired output (given by the labels in the training dataset), then we develop and run an algorithm that modifies the weights to reduce this error. For instance, the weights can be iteratively updated by stochastic

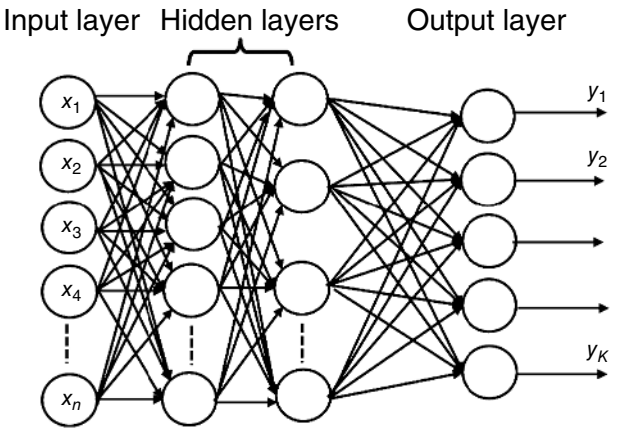


Figure 1.12 A feedforward multilayer neural network.

gradient descent (SGD) algorithm. The SGD algorithm requires computation of the gradient of the objective function with respect to the weights of all layers.

Backpropagation (Rumelhart et al. 1986) is the key to training deep neural networks. The idea behind backpropagation is the chain rule of derivatives. The derivative (or gradient) of the objective function with respect to the input of a layer can be computed by working backward from the derivative with respect to the output of the layer. The backpropagation can be applied repeatedly to propagate gradients through all layers, starting from the output (where the network produces its prediction), all the way to the input (where the external input is fed). Once these gradients have been computed, it is straightforward to compute the gradients with respect to the weights of each layer, and thus to update the parameters.

Instead of using a threshold function Eq. (1.1b) for non-linear activation, modern deep neural networks typically adopt various differentiable functions for smoother non-linearity or efficient derivative computation. The most popular activation function is the rectified linear unit (ReLU), defined as

$$g(z) = ReLU(z) = \begin{cases} z & if\ z \geq 0 \\ 0 & if\ z < 0 \end{cases} \tag{1.2}$$

Another important activation function is the sigmoid function, defined as

$$g(z) = \sigma(z) = \frac{1}{1 + \exp(-z)} \tag{1.3}$$

In this book, **Chapter 4** (Basics of Neural Networks) provides a general treatment of multilayer neural networks and their training by backpropagation. **Chapter 5** (Practical Considerations in Neural Networks) addresses important practical issues, including overfitting, k-fold cross-validation, regularization, weight initialization, batch gradient descent, Adam optimization, and so on.

1.4.3 Convolutional Neural Networks (CNNs)

In the 2010s, deep learning experienced a resurgence of interest and rapid development. This was mainly due to the availability of large amounts of data, powerful computing resources, and the development of new network architectures such as CNNs and recurrent neural networks (RNNs). These new models and architectures enabled the development of deep learning applications such as natural language processing (NLP), computer vision, autonomous driving, and many more.

CNNs are a type of neural network mainly used for image and video recognition, as well as for NLP later. The structure of a CNN is designed to take advantage of the 2D structure of an input, such as an image. First, local groups of values in an image are often highly correlated. Second, the local statistics of images are invariant to location. Thus, there are four ingredients in CNNs: local (or sparse) connections, shared weights, pooling operations, and many layers.

A CNN consists of an input layer, an output layer, and multiple hidden layers in between. The hidden layers typically include a combination of convolutional layers, activation layers, and pooling layers. Figure 1.13 illustrates a typical CNN architecture for image classification.

Convolutional layers. These layers are responsible for detecting features in the input, such as edges or textures. In a convolutional layer, a small matrix called a kernel or filter is used to scan the input image, and for each location, the kernel performs a dot product between its values and the

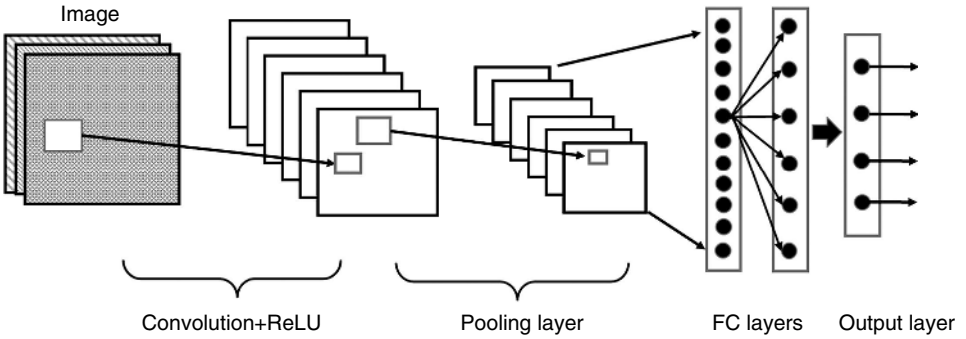


Figure 1.13 A simple CNN architecture for image classification. Note: more combinations [convolutional layer + ReLU + pooling layer] can be inserted right before the first FC layer.

input values. The output of this operation is called a feature map. By using multiple filters (six filters shown in Figure 1.13), a convolutional layer can deliver multiple feature maps to the next layer.

Activation layers. These layers introduce non-linearity to the network by applying an activation function, e.g. ReLU or sigmoid (not shown in Figure 1.13).

Pooling layers. These layers reduce the dimensionality of the feature maps produced by the convolutional layers. They achieve this by applying a function such as max-pooling or average-pooling to small regions of the feature map. This has the effect of retaining only the most important information in the feature map and making the network more robust to small changes in the input.

Fully connected layers. These layers are connected to all the neurons of the previous layers. They are used to interpret the features produced by the convolutional and pooling layers and produce an output.

Output layer. The output layer produces the final prediction for the input image, for instance, a probability distribution for an image classification task.

To develop deep learning models in an efficient way, we use deep learning frameworks (e.g. PyTorch, TensorFlow) to construct and train the models. **Chapter 6** (Introduction to PyTorch) gives an introduction to PyTorch, a popular framework. **Chapter 7** (CNNs) presents the basics of CNNs, including mathematical descriptions, examples (e.g. LeNet-5), and implementations of CNNs in PyTorch for image classification. **Chapter 8** (Classic Architectures of CNNs) describes several important deep CNN models including AlexNet, VGG, Network-in-Network, GoogLeNet, and ResNet. **Chapter 9** (Object Detection) presents an object detection technique that is an application of CNNs in computer vision. **Chapter 10** (Introduction to Probabilistic Generative Models), **Chapter 11** (Generative Adversarial Networks), and **Chapter 12** (Diffusion Models) provide a comprehensive treatment of generative models and their applications for image generation implemented by CNNs.

1.4.4 Recurrent Neural Networks (RNNs)

RNNs are a family of neural networks that process sequential data, such as time series or natural language. They are different from traditional feedforward neural networks in that they have a "memory" that allows them to use the result from previous timesteps as part of the input for the current timestep.

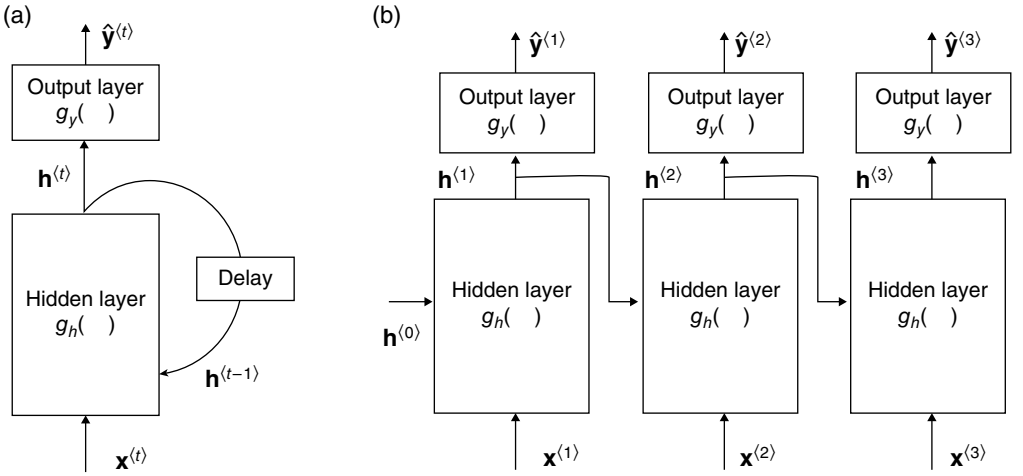


Figure 1.14 The architecture of a basic recurrent neural network: (a) RNN and (b) the unfolded representation of RNN.

The architecture of a basic RNN is illustrated in Figure 1.14(a). The RNN generates a prediction sequence $\left\{\hat{\mathbf{y}}^{\langle t \rangle}, t = 1, 2, 3, ...\right\}$ for the input sequence $\{\mathbf{x}^{\langle t \rangle}, t = 1, 2, 3, ...\}$. In general, one element in either sequence, $\mathbf{x}^{\langle t \rangle}$ or $\hat{\mathbf{y}}^{\langle t \rangle}$, is a vector. For example, in machine translation, the input sequence can be a sentence in one language while the prediction sequence is the corresponding sentence in another language. $\mathbf{x}^{\langle t \rangle}$ or $\hat{\mathbf{y}}^{\langle t \rangle}$ is the vector representation of a word in its language. The RNN processes one element at a time, maintaining in the hidden units a state vector $\mathbf{h}^{\langle t \rangle}$ that implicitly contains information about the history of the past elements of the sequence. The hidden state vector is updated at each timestep based on the input and the previous hidden state. The hidden state vector can be thought of as a summary of the information that the network has seen up to that point in the sequence. Based on this hidden state vector, the output layer generates the prediction depending on a specific task.

RNNs can be unfolded in time as in Figure 1.14(b) and viewed as very deep feedforward networks in which all the layers share the same weights. The major problem of training RNNs is that the backpropagated gradients either grow or shrink at each timestep, so over many timesteps they may explode or vanish. This makes it difficult to learn long-term dependencies. In the late 1990s, a variant of RNN called long short-term memory (LSTM) was proposed to address the problem of exploding or vanishing gradient. This made it possible for the RNNs to be trained on much longer sequences for more complex tasks.

In recent years, RNNs have seen a resurgence of interest, particularly with the advent of deep learning and the availability of large amounts of data and computational resources. With the help of these advancements, RNNs are now used in a wide range of applications, including NLP, speech recognition, and computer vision. Furthermore, the attention mechanism has been proposed and applied to RNNs, which further boosts the performance of RNNs on various NLP tasks such as machine translation, text summarization, question answering, etc.

In this book, **Chapter 13** (Word Embedding) presents the principle of how to represent each word in a language by a numerical vector. Understanding the purpose of word embedding will greatly help us in studying transformers. **Chapter 14** (Recurrent Neural Networks) presents the

fundamentals of RNNs, including architecture of RNNs, gradient backpropagation through time (exploding/vanishing gradient), LSTM, gated recurrent units (GRU), deep RNNs, attention in RNNs, and sequence-to-sequence learning. **Chapter 15** (Transformer) describes *transformers*, which are a powerful, generic deep learning model based on an attention mechanism and widely used for various applications in NLP.

1.4.5 Reinforcement Learning

As mentioned in Section 1.2.3, RL is a type of machine learning where an agent learns to make decisions and apply an action to an environment, as shown in Figure 1.6. The agent receives feedback in the form of rewards or penalties from the environment, and uses this feedback to adjust its decision-making process over time. The goal of the agent is to maximize the cumulative reward it receives over time. The agent learns by experimenting with different actions in the environment and observing the consequences. This process is known as trial-and-error learning or experience-based learning.

In RL, the agent decision-making process is typically modeled as a Markov decision process (MDP), where the agent's action at each timestep depends only on the current state of the environment. There are two main types of RL algorithms: *value-based* and *policy-based*. Value-based methods estimate the value of different states or actions, and then use this information to guide the agent decision-making. Policy-based methods, on the other hand, learn to directly select actions based on the current state of the environment.

The success of RL in playing games such as Go and Atari games, and its application in robotics, self-driving cars, and other real-world problems has led to an increased interest in the field. In recent years, RL has been combined with other fields such as computer vision, NLP, and control systems to solve more complex problems and improve the performance of the agent.

In this book, **Chapter 16** (Introduction to Reinforcement Learning) describes the basic concepts of MDP and how to solve an MDP problem by value-based methods, assuming that the mathematical model of the environment is known. This chapter provides a foundation for RL and is a prerequisite for the subsequent chapters. **Chapter 17** (Deep Q-learning) presents Q-learning as a practical and powerful value-based RL algorithm, which can be implemented by deep neural networks. **Chapter 18** treats policy-based methods, which learn the policy function directly.

1.5 Resources for Deep Learning

Python is one of the most popular computer languages for machine learning. Unlike MATLAB, Python is an open-source language and is free to use. Since Python and machine learning packages have been updated frequently by the community, one should pay attention to any upgrade issues or any deprecated functions while trying to run the code provided by the book.

In addition to downloading and setting up Python platform on local computers, one can choose to use a cloud computing service. For example, Google provides a free online Jupyter notebook service, called Colaboratory (https://colab.research.google.com/), or "Colab" for short. Colab allows one to write and execute arbitrary Python code through the browser, and is especially well suitable for machine learning, data analysis, and education, with access to computing resources including GPUs.

1.5.1 Frameworks

Although it is essential for a reader to understand the mathematical foundations of machine learning, machine learning frameworks play a vital role in practice. A machine learning framework is a library, interface, or tool that enables developers to build models more easily and rapidly. It allows the developers to create a particular ML application without starting from scratch. To use a framework, we need to install and activate the corresponding packages in the Anaconda3 environment and *import* the packages typically at the beginning of a Python program.

There are several popular frameworks for deep learning, each with its own strengths and weaknesses. Some of the most used frameworks are:

TensorFlow. Developed by Google, TensorFlow is an open-source library that is widely used for both research and production. It is designed to be flexible and extensible, allowing for easy experimentation with new models and architectures. It also has a large community and a wealth of resources and tutorials available.

PyTorch. Developed by Facebook, PyTorch is an open-source library that is similar to TensorFlow in many ways. It is known for its simplicity and ease of use, making it a popular choice for researchers and developers. PyTorch also has a dynamic computational graph that allows for easy debugging and modification of the model.

Caffe. Developed by Berkeley Vision and Learning Center (BVLC), Caffe is a deep learning framework that is focused on speed and efficiency. It is particularly well-suited for image and video processing tasks, and it includes pre-trained models for a variety of computer vision tasks.

CNTK. Developed by Microsoft, cognitive toolkit (CNTK) is an open-source deep learning toolkit that is designed to be efficient and flexible. It supports a wide range of models and architectures, and it also includes tools for distributed training and data pre-processing.

Theano. Developed by Montreal Institute for Learning Algorithms (MILA), Theano is an open-source library focusing on efficiency and performance. It is designed to make it easy to perform large-scale numerical computations on both central process unit (CPUs) and GPUs.

Keras. Developed by Francois Chollet, Keras is a high-level deep learning framework that is written in Python. It is designed to be simple and easy to use, making it a popular choice for beginners. Keras can run on top of TensorFlow, CNTK, or Theano.

Apache MXNet. Developed by the Apache Software Foundation, Apache MXNet is designed to be highly scalable and can handle large datasets, making it well-suited for distributed and parallel computing. The MXNet library is portable and lightweight. It is accelerated with the NVIDIA Pascal™ GPUs and scales across multiple GPUs and multiple nodes, allowing you to train models faster. It also allows you to define, train, and deploy deep neural networks on a wide array of devices, from cloud infrastructure to mobile devices.

1.5.2 Resources for Studying Deep Learning

There are many resources available for learning deep learning, including online tutorials, courses, and books. Some examples are listed below.

Online courses. Websites such as *Coursera*, *edX*, and *Udemy* offer a wide variety of deep learning courses. These courses cover a wide range of topics, from beginner-friendly introductions to more advanced topics such as computer vision and NLP. The courses offered by Andrew Ng through Coursera are most popular and highly recommended for beginners. Many universities also offer online courses in AI related topics.

The classic 10-lecture course (Silver 2015), taught by David Silver, was recorded in 2015 and remains a popular resource for anyone wanting to understand the fundamentals of RL. The videos and slides are available online.

Books. There are many books available on deep learning, for either beginners or advanced practitioners. The first comprehensive and authoritative book on deep learning is the book "*Deep Learning*" (Goodfellow et al. 2016), which covers deep learning topics up to 2016. The book "*Pattern Recognition and Machine Learning*" (Bishop 2006) treats machine learning from a perspective of math in depth, and its author (Bishop) later published a book "*Deep Learning: Foundations and Concepts*" (Bishop and Bishop 2024) with a focus on deep learning.

The book "*Dive into Deep Learning*" (Zhang et al. 2022) is an excellent textbook on deep learning with detailed code, math, and discussions. Examples in the book are implemented with different frameworks such as PyTorch, NumPy/MXNet, and TensorFlow. This book is available online in both html and PDF formats. Other popular books include, but are not limited to, "*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*" (Geron 2017 and "*Neural Networks and Deep Learning, a Textbook*" (Aggarwal 2018).
However, the topic of RL was not sufficiently covered by the above-mentioned books. A classical textbook for RL is "*Reinforcement Learning: An Introduction*" (Sutton and Barto 2018). To get hands-on experience, one can refer to the book "*Deep Reinforcement Learning Hands-On*" (Lapan 2018).

Blogs and websites. There are many blogs and websites dedicated to deep learning and machine learning, such as TensorFlow, Keras, and PyTorch, which provide tutorials, tips, and updates on the latest developments in the field.

Towards Data Science (TDS) is an online platform that aims to share knowledge and best practices for data science and machine learning. It features a wide range of content, including articles, tutorials, and interviews, all of which are written by data scientists and machine learning experts from around the world. TDS covers a wide range of topics related to data science and machine learning, including data visualization, NLP, deep learning, and big data. The articles on TDS are written in a clear and concise manner, making them accessible to a wide range of readers, including both beginners and experienced data scientists.
Medium is another online platform that allows individuals and organizations to share articles and stories on a wide range of topics, including machine learning.
GitHub is a popular platform for software developers to share their projects. The platform makes it easy for developers to discover and contribute to open-source projects, and allows them to share their own projects with others. This has led to the creation of a vast ecosystem of open-source software, with millions of projects hosted on GitHub. Many typical machine learning projects can be found on *GitHub*.

Conferences and workshops. Conferences such as NeurIPS (Neural Information Processing Systems), ICML (International Conference on Machine Learning), and ICLR (International Conference on Learning Representations), and workshops such as the Google AI Residency, OpenAI, and fast.ai provide opportunities to learn from experts in the field and network with other practitioners.

Research papers. Reading research papers is an excellent way to gain a deeper understanding of the technical details and latest developments in deep learning. Websites such as arXiv and Google Scholar provide access to a wide variety of research papers in this field.

Exercises

1.1 Define Machine Learning in your own words.

1.2 Identify whether the following task can be addressed by a supervised learning algorithm or an unsupervised learning algorithm?

1) Given email labeled as spam/not spam, learn a spam filter.
2) Given a set of news articles found on the web, group them into sets of articles about the same stories.
3) Given a database of customer data, automatically discover market segments and group customers into different market segments.
4) Given a dataset of patients diagnosed as either having diabetes or not, learn to classify new patients as having diabetes or not.
5) Given a database of used car sales, estimate the market price of a particular used car.

1.3 What are the two most common types of supervised learning? Can you give some examples for each of them?

1.4 Do the following installations on your computer:

1) Install Anaconda
2) Install Jupyter notebook (possible steps: Run Anaconda prompt; Create an environment; Activate the environment; Install ipython, jupyter, ipykernel)
3) Install PyTorch (at the environment-activated prompt)
4) Install other packages: numpy, scikit-learn, pandas, matplotlib, statsmodels, seaborn, torchvision, scikit-image.
5) Write a Python program and run it in your Jupyter notebook. The program should print the version of all packages you installed, including torch, numpy, scipy, matplotlib, statsmodels, sklearn, pandas, seaborn,
6) Write a Python program that uses NumPy, creates two random 100×100 arrays, and adds them together in two different ways: the first by using a double for-loop, and the second by using the NumPy "+" operator. Time how long each method takes to add the arrays.

1.5 A logic OR gate is implemented by a single perceptron with three inputs in Figure 1.11. In similar way, implement: 1) AND gate; 2) NAND gate; 3) NOR gate; 4) NOT gate, and find the corresponding parameters. Can we implement an XOR gate with such a single perceptron? Why?

1.6 Sketch the graph for functions: $\sigma(z) = \dfrac{1}{1 + \exp(-z)}$, and $\tanh(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$.

1.7 Compute the derivative of sigmoid function $\sigma(z) = \dfrac{1}{1 + \exp(-z)}$, and verify

$$\sigma(-z) = 1 - \sigma(z)$$

$$\frac{d}{dz}\sigma(z) = \sigma(z)(1 - \sigma(z))$$

References

Aggarwal, C.C. (2018). *Neural Networks and Deep Learning, a Textbook*. Springer.

Bishop, C.M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Bishop, C.M. and Bishop, H. (2024). *Deep Learning: Foundations and Concepts*. Springer.

Geron, A. (2017). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Lapan, M. (2018). *Deep Reinforcement Learning Hands-On*. Packt.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* 65 (6): 386–408. https://doi.org/10.1037/h0042519.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986). Learning Representations by Back-Propagating Errors. *Nature* 323: 9.

Silver, D. (2015). A course on introduction to reinforcement learning. https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver

Sutton, R.S. and Barto, A.G. (2018). *Reinforcement Learning: An Introduction*, 2e. MIT Press.

Zhang, A., Lipton, Z.C., Li, M., and Smola, A.J. (2022). Dive into deep learning. https://d2l.ai/