Chapter 1

Introduction to Deep Learning

Learning Objectives

In this chapter we provide an introduction.

- Types of machine learning
- Data representation in machine learning
- Basic concepts on Datasets: data examples, features, label
- Three data sets: training set, testing set, and validation set
- History of deep learning
- Resources for learning deep learning
- Roadmap of this book

1.1 Introduction

Over the last a few 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*, OpenAI Chat-GPT (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, OpenAI Chat-GPT can write codes for you. If you want a paragraph to be translated into a different language, you can ask OpenAI Chat-GPT to do it. If you are a Go player and want to improve your skill, you can ask AlphaGo (an AI program) (https://www.deepmind.com/research/highlighted-research/alphago) for the best move.

Chat-GPT 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 Chat-GPT 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 that exhibit a level of knowledge and understanding that is similar to that of a human.

AlphaGo is a computer program that defeats Go world champions in different global arenas and became the strongest Go player in history. The model combines advanced search tree with deep neural networks and is trained through reinforcement learning.

The goal of this book is to present the fundamentals of neural networks and deep learning. Deep learning is the engine of AI while neural networks are the major key components in deep learning. In fact, deep learning was first introduced to machine learning in 1980s. The rapid development of deep learning in recent years is a result of the significant advances in computer technology (e.g., GPU) and the availability of large amount of data. The graphic process units (GPUs), designed originally for graph visualization, provide a powerful computation on matrix data, which is a major form of the data structure in deep learning. The combination of the powerful computation and large memory hardware makes it possible to train large neural networks with millions of parameters. Data in deep learning is analogous to the fuel of the rocket engine. Without sufficient training data, there was no way to find the *good* parameters for the neural networks. Over the past two decades, massive amount of data (e.g., images, videos, social network data) have been generated through the internets and mobile electronic devices.

We begin by introducing some necessary terminology and a general formulation of machine learning. Then we present a brief overview of deep learning and how the key components in the knowledge of deep learning are linked to the chapters in this book. In this chapter, we also provide some useful resources as the supplementary to this book.

1.2 Types of Machine learning

The relationship of three terms, *Artificial Intelligence*, *Machine Learning*, and *Deep Learning* can be illustrated by Fig.1, where artificial intelligence refers to 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 which performs machine learning inspired by the neural networks in human's brain.

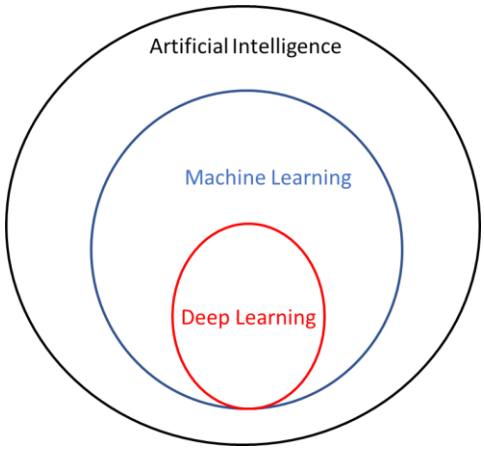


Fig.1 Artificial intelligence, machine learning and deep learning

Although the phrase "machine learning" has become more and more frequently used in every aspect of our lives, it is challenging to give a perfect definition for machine learning. Even among machine learning practitioners, there are different definitions for machine learning. In 1950s, Arthur Samuel

defined machine learning as the field of study that gives computers the ability to learn without being explicitly programmed. A recently well-accepted definition 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.*

For example, an email spam filter is a computer program that predicts an incoming email as spam or not spam, based on many examples 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 human as spam or not spam. 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 the future situations. By generalizing, the machine can recognize the similarity between the data seen and the data unseen.

When we discuss or develop a specific machine learning system, it is helpful or necessary to identify the type of the system. Machine learning systems can be classified according to the amount and type of supervision available during training. In this section, we present three basic types of machine learning systems: supervised learning, unsupervised learning, and reinforcement learning, though semi-supervised learning and self-supervised learning are attracting our attentions recently.

1.2.1 Supervised learning

Supervise learning is the most common type of machine learning. Literally, it means that the learning process is supervised by a labelled dataset. Specifically, we train a **model** based on a **training dataset** including the sample points consisting of both the input **features** and the output **labels**. The model usually has many parameters with randomly initial values. The objective of learning process is to search for the best values of the parameters so that the outputs of the model are *close* to the true labels for the given input features. Furthermore, a successful model should have a capability of **generalization**: the model should produce sensible outputs for the input features that were not encountered in the training dataset.

In supervised learning settings, the training dataset is a collection of samples of both the input features and the output labels, denoted as $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}), i = 1,2, \dots, m\}$. The superscript (i) indicates the data point as the *ith* samples, *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 (or target or answer 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 inputs and labels, so that it can predict the label for a new input $\mathbf{x}$ without a label given. In general, $\mathbf{x}^{(i)}$ is a vector or high-dimensional matrix while $\mathbf{y}^{(i)}$ is a scalar or vector. Fig.2 illustrates a simple dataset with the input features (x_1, x_2) and the output label y. In many applications the number of input features is much larger. For example, in

computer vision, a 224×224 RGB (Red-Green-Blue) color image has 224×224×3=150,528 features.

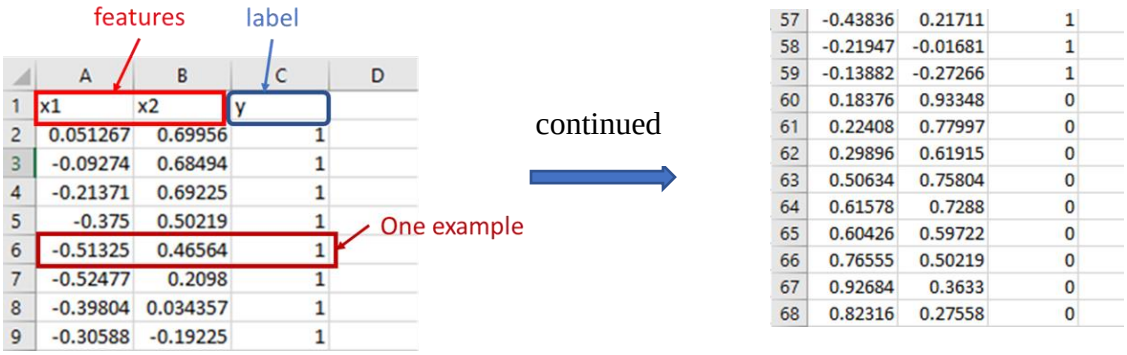


Fig.2 A dataset with two features and one label

In supervised learning, the training process is to learn a model from a labeled training dataset so that the resulting model can effectively predict the label for a new input sample, as illustrated in Fig.3. The training process typically consists of many iterations on updating the parameters of the model.

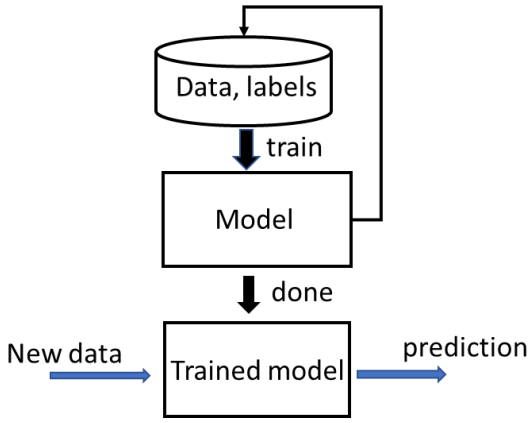
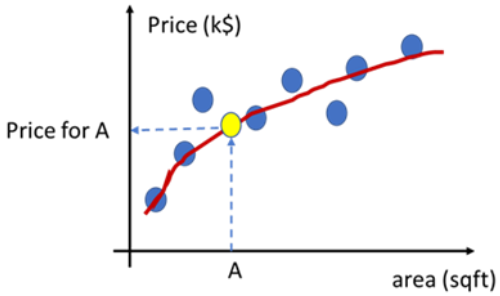


Fig.3 Supervised learning

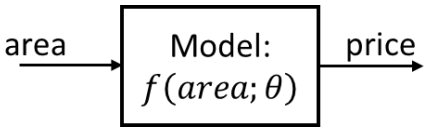
Supervised learning can be further categorized into regression and classification. In a regression task, the labels are continuously valued. In other words, we are trying to map input variables to some continuous function. In a classification task, on the other hand the labels are discrete. We are trying to predict 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 the training data set including information about houses (e.g., area, number of bedrooms, age, number of floors) and their market prices. The information about houses is considered as the input features while the market prices are the labels. A trained model is expected to predict (or estimate) the price for a house based on its information (in general not present in the training set). For a purpose of visualization, we consider only one feature of the house – area. The example points (area, price) can be plotted as blue dots in Fig.4 (a). To predict the price for a house with a certain area, the objective of regression is to find a

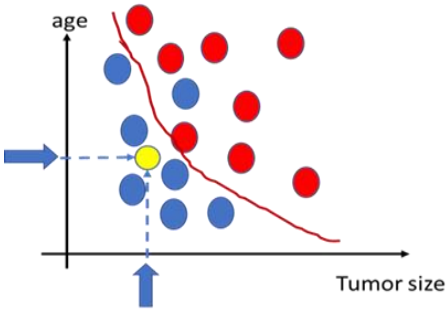
curve (red) to fit the data examples according to a criterion, e.g. minimal mean squared error (MSE). Based on the fitting curve, we can predict the price for a house with any area A, shown by the dash arrow line. The model can be mathematically described by a function $f(area; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the model parameter. The output of the function is the house price, shown in Fig.4(b).



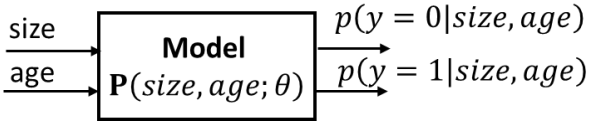
(a) regression



(b) model for regression



(c) classification



(d) model for classification

Fig.4 Regression vs. Classification

An example of classification is illustrated in Fig.4(c), which is the classification of breast cancer, given a training data set including many patients' features and labels—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 person given her features. To visualize the concept, we consider two features: age and tumor size. The training data samples are plotted as blue dots for benign samples and red dots for malignant samples. After learning from the training data set, the model will be able to predict the category for any pair (tumor size, age). Thus, a decision boundary (indicated by the red curve), which separates two types of breast cancers, exists for a trained model. In the case of Fig.4(c), the new data sample indicated by the yellow dot is classified as benign by the model. In practice, to predict the discrete label, the model estimates the conditional probability of each category associated with the input, as shown in Fig.4(d), where $p(y = 0|size, age)$ represents the probability of y=0 given a pair of values (size, age). The category with the largest probability at the output is predicted.

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

1.2.2 Unsupervised learning

In unsupervised learning, the training data is not labeled. A machine learning algorithm is expected to derive the structure pattern of the data. For instance, the structure can be obtained by clustering the data samples based on relationships among the input features. The model tries to identify the similarity between data samples so that the samples have something in common are categorized together. Fig.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 yellow and purple dash circles.

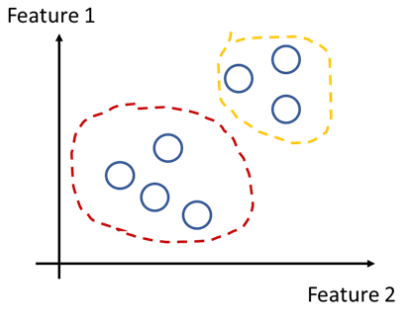


Fig.5 Unsupervised learning: clustering

As examples, 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 into groups that are somehow similar or related by different variables, such as lifespan, location, roles, and so on.
b) Computer clusters: consider a large data center with a large number of computer machines. We try to organize the machines into different clusters (or groups) based on the role of each machine, so that the data center work efficiently.
c) Social network analysis: given the activities of a person in social networks such as emails, Facebook, Instagram, WeChat, we try to analyze the nature of his/her social groups.
d) Market segmentation: given a huge database of customer information, we try to group customer into different market segments so that we can sell products more efficiently.

Visualization and dimensionality reduction also belong to unsupervised learning. These algorithms try to transform higher dimensional data set 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.

1.2.3 Reinforcement learning

In reinforcement learning, we want to develop an **agent** that can intelligently interact with an **environment**, as illustrated in Fig.6. In each interaction, the agent selects an **action** from a set of actions, and performs the selected action to the environment. As the consequence of the action, 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 bast strategy, called a **policy**, to get the most reward over time. A policy defines what action the agent should take in a given situation (e.g. state).

An example of reinforcement learning is the AlphaGo program, which defeated the world champion Ke Jie at the game of Go in 2017.

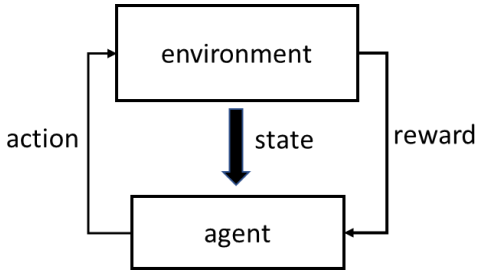


Fig.6 Reinforcement learning

1.3 Data Representation in Machine Learning

1.3.1 Tensor

In machine learning, data are usually organized as a data structure, called tensor, which is a generalization of vectors and matrices and is easily understood as a multidimensional array. A vector is a one-dimensional (1D) tensor, and a matrix is a two-dimensional (2D) tensor. One dimension is associated with one **axis**. Higher dimensional tensors can be defined with more axes. 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 size 4 and the second axis size 2, and thus its shape is denoted as size (4,2). The **shape** of a tensor is a tuple of integers that describe how many elements of the tensor have along each axis.

A set of *m* datapoint samples can be represented by a single tensor. For instance, *m* samples of vector can be represented by a tensor with a shape of (*m*, *n*), where *n* is 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 color value for one pixel, illustrated in Fig.7), H is the height, and W is the width. A set of *m* images can be represented by a 4D tensor with a shape of (*m*, 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.

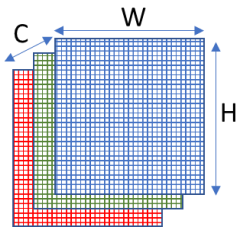


Fig.7 Data structure of an image with three colors (red, green, blue)

1.3.2 Datasets: training, validation, and testing

In machine learning, data are usually organized in datasets which are stored in one or multiple files. To develop a machine learning model, we need three types of datasets in terms of the purpose: **training set**, **validation set**, and **testing set**. Ideally, all data sets are generated in the same statistical condition and have the same format. In other words, all three data sets have the same statistical properties.

In general, the process of developing a machine learning product consists of three steps as shown in Fig.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 for the final product.

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

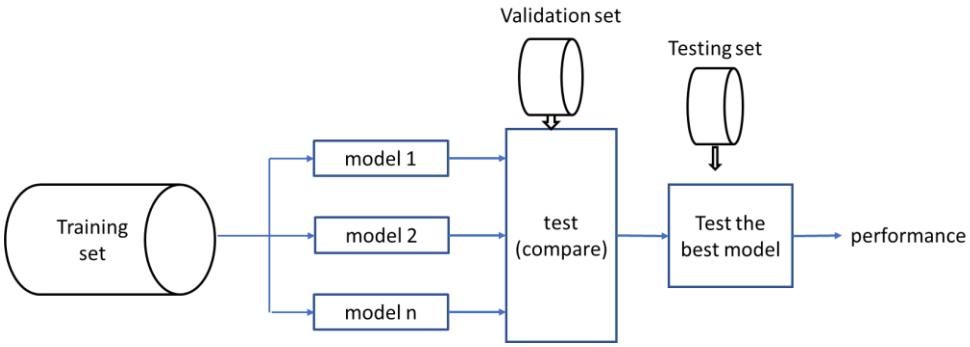


Fig.8 Development of a machine learning product

In many situations, obtaining a large amount of data (especially labeled data) is expensive. This leads to limited amount data available. Thus, it is essential to use the available data efficiently. The question is: given an entire data set, how do we split it into three sets? A basic guideline to create three data sets is illustrated by Fig.9, and described a s follows:

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. Practically, 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 test sets are expected to be reduced so that more data can be used for training. For example, for a size of 1,000,000 dataset, a reasonable partition would be 980,000 (98% training), 10,000 (1% validation), and 10,000 (1% test).

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 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 data set 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 data set with unbalanced classes.

We will see how this is implemented as we look at relevant examples in the proceeding chapters.

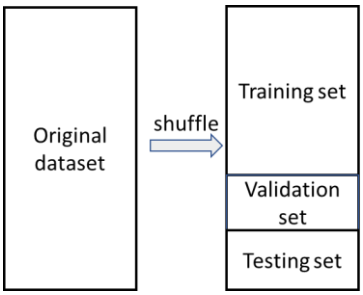


Fig.9 Generate training set, validation set, and testing set from the original dataset

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 data sets. Up to date, the repository maintains more than six hundred datasets as a service to the machine learning 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 find and publish 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 via Kaggle.

1.4 An Overview of Deep Learning

In this section we present an overview of deep learning, which highlights the most important advances in the history of deep learning. The purpose of this section is to provide a roadmap for this book, and motivate our readers.

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 artificial intelligence 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 that consisted 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 Fig.10, and defined by

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

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

where $x_i, i = 1,2 \ldots m$, are m input features, 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. This threshold function is called an activation function.

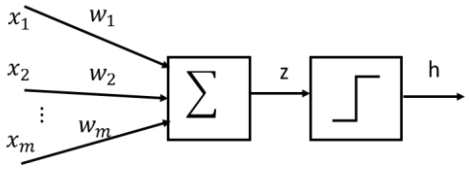
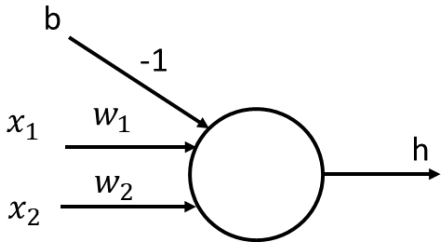


Fig.10 A neuron in perceptron

A simple example of perceptron is to implement a logic function OR gate with a single neuron, as shown in Fig.11, where the circle represents a neuron that includes both the adder and the activation function defined in (1.1), 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.



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

Fig. 11 The perceptron for OR gate

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 start point for us to understand the basic concepts of neural networks. **Chapter 3** treats linear regression corresponding to the adder in the neuron while **Chapter 4** is dedicated for logistic regression that is a neuron with a smooth activation function, such as sigmoid function.

1.4.2 Multi-layer 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 at the time. Despite these limitations, some researchers continued to develop more complex models, such as multilayer networks, which were capable of solving 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.

A feedforward multi-layer network is illustrated in Fig. 12. The network has an input layer that just stores the input data with no operations, two hidden layers, and an output layer. All nodes, except those in the input layer, are neurons with identical structure. Each layer may have multiple neurons. Each neuron receives signals from all 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). There is an adjustable parameter, called weight, associated with each connection. However, some practical neural networks (e.g., convolutional neural networks) are sparsely connected, and can be understood as special cases where most of connections between layers have weight zero.

The output of the output layer is the prediction of the network for the current input.

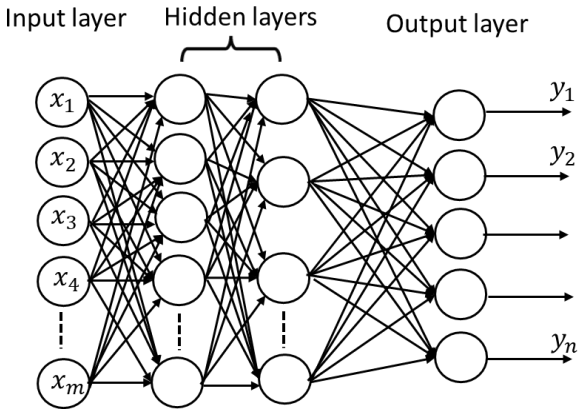


Fig.12 A feed-forward multi-layer neural network

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 develop and run an algorithm that modifies the weights to reduce this error. For instance, the weights can be iteratively updated by stochastic gradient descent (SGD) algorithm. The SGD algorithm requires the computation of the gradient of the objective function with respect to the weights of all layers.

Backpropagation is an important breakthrough for 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 backwards 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.

Instead of using a threshold function (1.1b) for non-linear activation, modern deep neural networks typically adopt different activation functions for smoother non-linearities 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 sigmoid function, defined as

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

In this book, **Chapter 5** provides a general treatment of multilayer neural networks and training by backpropagation. **Chapter 6** treats important practical issues in real-world projects, 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 a rapid development in new techniques, architectures and models. This was mainly due to the availability of large amounts of data, powerful computing resources and the development of new training algorithms such as backpropagation, convolutional neural networks and recurrent neural networks. These new models and architectures enabled the development of deep learning applications such as natural language processing, computer vision, autonomous driving and many more.

Convolutional neural networks (CNNs) are a type of neural networks used for image and video recognition, as well as natural language processing. 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. As the result, there are four key ideas behind CNNs: local (or sparse) connections, shared weights, pooling and the use of many layers.

A CNN consist 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. Fig.13 illustrates a typical CNN architecture for image classification.

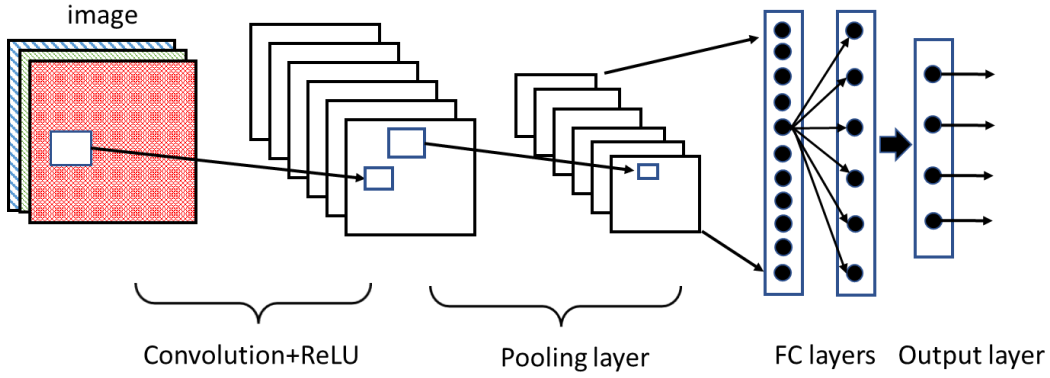


Fig.13 A simple CNN architecture for image classification. Note: more combinations [convolutional layer+ReLU+pooling layer] can be inserted right before the first FC layer.

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 input values. The output of this operation is called a feature map. By using multiple filters (6 filters shown in Fig.13), a convolutional layer can detect multiple features in the input.

Activation Layers: These layers introduce non-linearity to the network by applying an activation function such as ReLU (Rectified Linear Unit) or sigmoid.

Pooling Layers: These layers reduce the dimensionality of the feature maps produced by the convolutional layers. They do 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, such as the class label for an image classification task.

In the ImageNet competition in 2012, deep CNNs achieved spectacular success when applied to a data set of about a million images that contained 1000 different classes. Since then, CNNs have been the dominant approach for almost all recognition and detection tasks and approach or exceed human performance on some tasks.

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 7** gives an introduction to PyTorch, a popular framework. **Chapter 8** presents the basics of CNNs, including mathematical descriptions, examples (e.g. LeNet-5), and implementations of CNNs in PyTorch for image classification. **Chapter 9** describes several important advanced deep CNN architectures including AlexNet, VGG, Network-in-Network, GoogLeNet, and ResNet. **Chapter 11** and **Chapter 12** present two important applications of CNNs in computer vision: object detection and generative adversarial nets (GANs).

1.4.4 Recurrent neural networks (RNNs)

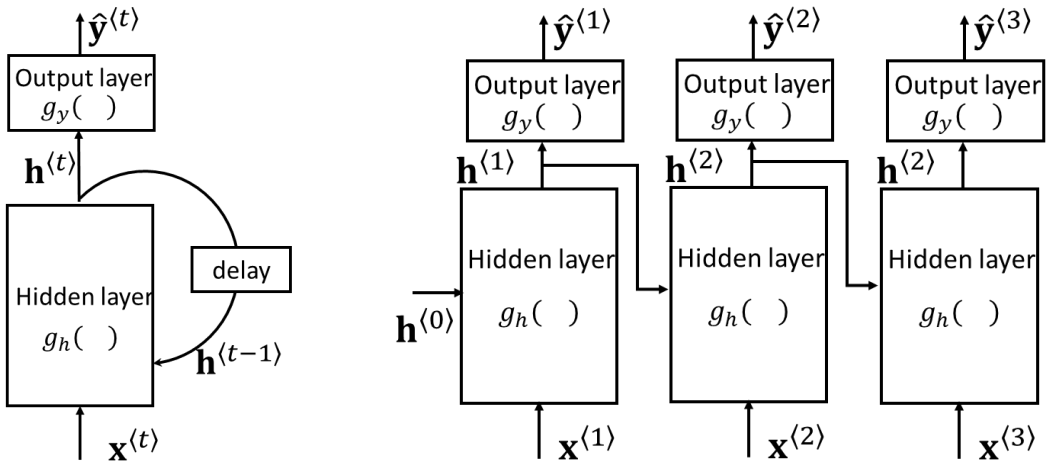
Recurrent neural networks (RNNs) are a type of neural networks that are designed to process sequential data, such as time series or natural language. They are different from traditional feedforward neural networks in that they have a "memory" that allows them to use information from previous time steps to process the input at the current time step.

The architecture of a basic RNN is illustrated in Fig.14(a). The RNN generates a prediction sequence $\{\hat{\mathbf{y}}^{\langle t \rangle}, t = 1,2,3, \ldots\}$ for the input sequence $\{\mathbf{x}^{\langle t \rangle}, t = 1,2,3, \ldots\}$. 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 an input sequence one element at a time, maintaining in their hidden units a state vector $\mathbf{h}^{\langle t \rangle}$ that implicitly contains information about the history of all the past elements of the sequence. The hidden state vector is updated at each time step 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 application. For simplicity, both the hidden layer and the output layer can be understood as a neuron.

RNNs can be unfolded in time as in Fig.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 time step, so over many time steps they typically 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 of data and 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 natural language processing, speech recognition, and computer vision. Furthermore, attention mechanism has been proposed and applied on RNNs which further boosts the performance of RNNs on various NLP tasks such as machine translation, text summarization, question answering etc.



(a) RNN (b) The unfolded representation of RNN

Fig.14 The architecture of a basic recurrent neural network

In this book, **Chapter 13** presents the fundamentals of RNNs, including language models for RNNs, gradient backpropagation through time (exploding/vanishing gradient), LSTM, gated recurrent units (GRU), and deep RNNs. **Chapter 15** introduces attention mechanism to RNNs. This results in a powerful generic deep learning model, *transformer*, which is widely used for various applications in natural language processing. **Chapter 14** covers word embedding, which explains how to generate the efficient representation of each word in a language for the input sequence of RNNs.

1.4.5 Reinforcement learning

Reinforcement learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment, as shown in Fig.6. The agent receives feedback in the form of rewards or penalties for its actions, 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.

Reinforcement learning is based on the idea of an agent learning to make decisions by trial and error, much like how animals or humans learn. 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 reinforcement learning, the agent's decision-making process is typically modeled as a Markov decision process (MDP), where the agent's actions at each time step depend only on the current state of the environment and the agent's previous actions. 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's decision-making. Policy-based methods, on the other hand, learn to directly select actions based on the current state of the environment.

The field of RL has grown significantly with the advancements in deep learning, leading to the development of new algorithms such as deep Q-networks (DQN) and the actor-critic method. 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, natural language processing and control systems to solve more complex problems and improve the performance of the agent.

In this book, **Chapter 16** 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. **Chapter 17** presents the basic RL algorithms based on tabular approaches for the tasks with a small size of state space. **Chapter 18** handles RL tasks with large state space by applying neural networks to approximate the value function, leading to powerful RL algorithms such as deep Q-learning. **Chapter 19** treats policy-based methods, which learn the policy function directly using neural networks.

1.5 Resources for Deep Learning

Python is one of the most popular computer languages for machine learning development. Unlike Matlab, Python is an open-source language and free to use. In this book, all examples or projects are run by Python 3.6 and relevant packages. Anaconda3 with Jupyter Notebook was downloaded in 2019, and serves as a design environment platform for this book (https://www.anaconda.com/). 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 in your own computer, you can choose to use cloud computing service. Google provides a free online Jupyter notebook service, called Colaboratory (https://colab.research.google.com/), or “Colab” for short. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to 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 environment Anaconda3 and *import* the packages typically in the beginning of a Python program.

There are several popular frameworks for deep learning, each with their own strengths and weaknesses. Some of the most commonly 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 which 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, 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 that is focused on efficiency and performance. It is designed to make it easy to perform large-scale numerical computations on both 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. It 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's 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.

These frameworks are all actively maintained and developed, and new ones are constantly emerging. The choice of framework depends on the specific requirements of the project and the preferences of the developer or researcher.

1.5.2 Resources for studying deep learning

There are many resources available for learning deep learning, including online tutorials, courses, and books. Some popular options include:

Online courses: Websites such as *Coursera*, *edX*, and *Udemy* offer a wide variety of deep learning courses. These courses cover a range of topics, from beginner-friendly introductions to more

advanced topics such as computer vision and natural language processing. The courses offered by Andrew Ng through Coursera are most popular and highly recommended for beginners.

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, both for beginners and advanced practitioners. The first comprehensive and authoritative book on deep learning is the book "*Deep Learning*" (Goodfellow, 2016), which covers a broad range of deep learning topics in deep learning up to 2016. The book "*Pattern Recognition and Machine Learning*" (Bishop, 2006) treats machine learning from a perspective of math in depth.

The book "*Dive into Deep Learning*" (Zhang, 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 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 reinforcement learning was not sufficiently covered by the above-mentioned books. A classical textbook for reinforcement learning is "*Reinforcement Learning: An Introduction*" (Sutton, 2018). To get hands-on experience, one can refer to the book "*Deep Reinforcement Learning Hands-On*" (Lapan, 12018).

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, natural language processing, 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 typically machine learning projects can be found in *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 an opportunity 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.

It's important to note that deep learning is a vast field, and it's impossible to learn everything at once. Start with the basics and gradually move on to more advanced topics. Also, practical experience is very important in deep learning, so it's recommended to implement models and experiment with them.

References

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

Bishop, C. M. (2006). Pattern recognition and machine learning. springer.

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

Goodfellow, I., Bengio, Y., & 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

Rosenblatt, F. The Perceptron — A Perceiving and Recognizing Automaton. Tech. Rep. 85-460-1 (Cornell Aeronautical Laboratory, 1957).

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, the second edition, MIT Press.

Zhang, A., Zachary C. Lipton, Mu Li, and Alexander J. Smola, (2022), Dive into Deep Learning, https://d2l.ai/

Exercises

1. Define Machine Learning in your own language.
2. Identify the following task can be addressed by supervised learning algorithm or 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 sells, estimate the market price of a particular used car.
3. What are the two most common types of supervised learning? Can you give some examples for each of them?
4. Do following installations in your computer if you have not done so:
    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.

   Example command could be

   (mypytorch) C:\Users\weido>conda install numpy

   Or

   (mypytorch) C:\Users\weido>pip install numpy

   where mypytorch is the name of the environment

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.

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

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

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