A Comprehensive Note on Machine Learning[1]

Jue Guo

January 4, 2024

Contents

I Basic Machine Learning **1**

1 Introduction **3**

1.1 What is Machine Learning . . . 3

1.2 Types of Learninig . . . 3

1.2.1 Supervised Learning . . . 3

1.2.2 Unsupervised Learning . . . 3

1.2.3 Semi-Supervised Learning . . . 4

1.2.4 Reinforcement Learning . . . 4

1.3 How Supervised Learning Works . . . 4

1.4 Why the Model Works on New Data . . . 7

2 Notation and Definition **9**

2.1 Notation . . . 9

2.1.1 Data Structure . . . 9

2.1.2 Capital Sigma Notation . . . 9

2.1.3 Capital Pi Notation . . . 10

2.1.4 Operations on Sets . . . 10

2.1.5 Operations on Vectors . . . 10

2.1.6 Functions . . . 11

2.1.7 Max and Arg Max . . . 12

2.1.8 Assignment Operator . . . 12

2.1.9 Derivative and Gradient . . . 13

2.2 Random Variable . . . 13

II Advance Machine Learning **15**

III Convolution Neural Networks **17**

IV Adversarial Attacks and Training **19**

3 Intriguing Properties of Neural Network **21**

3.1 Introduction 21

3.2 Framework 21

V Recurrent Neural Networks **23**

VI Transformers **25**

VII Artificial General Intelligence **27**

4 Continual Learning **29**

4.1 Introduction 29

4.2 Setup 30

4.2.1 Basic Formulation 30

4.2.2 Typical Scenairo 30

4.2.3 Evaluation Metrics 31

Preface

The primary purpose of this document is educational, specifically for the courses I teach (**CSE 474/574 Introduction to Machine Learning, CSE 455/555 Pattern Recognition, and CSE 676 Deep Learning**), as well as for my personal reference. A substantial portion of the material herein is directly referenced or adapted from established texts and sources, and is not claimed as original content. This document is intended as a supplementary teaching and learning resource and is not authorized for commercial use, redistribution, or sale without my explicit consent.

The majority of the material referenced comes from the following sources:

[1] Zhang, Aston, et al. Dive into deep learning. Cambridge University Press, 2023.

[2] Bishop, C. M., & Nasrabadi, N. M. (2006). Pattern recognition and machine learning (Vol. 4, No. 4, p. 738). New York: Springer.

[3] Hart, P. E., Stork, D. G., & Duda, R. O. (2000). Pattern classification. Hoboken: Wiley.

[4] Burkov, A. (2019). The hundred-page machine learning book (Vol. 1, p. 32). Quebec City, QC, Canada: Andriy Burkov.

[5] Burkov, A. (2020). Machine learning engineering (Vol. 1). Montreal, QC, Canada: True Positive Incorporated.

All other referenced materials and sources are cited in the bibliography section of this document. This compilation is intended to provide a comprehensive overview and guide for students and practitioners of machine learning, drawing upon a wide range of foundational and contemporary sources in the field.

Part I

Basic Machine Learning

Chapter 1

Introduction

1.1 What is Machine Learning

Machine learning is a subfield of computer science that is concerned with building algorithms which to be useful, rely on a collection of examples of some phenomenon.

- The examples come from nature, handcrafted by humans or generated by another algorithms.

Machine learning can also be defined as the process of solving a practical problem by

1. gathering a dataset
2. algorithmically building a statistical model based on the dataset.

The statistical model is assumed to be used somehow to solve the practical problem.

1.2 Types of Learninig

Learning can be **supervised**, **semi-supervised**, **unsupervised** and **reinforcement**.

1.2.1 Supervised Learning

In **supervised learning**, the **dataset** is the collection of **labeled examples** $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$.

- Each element $\mathbf{x}_i$ among N is called a **feature vector**. A feature vector is a vector in which each dimension $j = 1, ..., D$ contains a value that describes the example somehow. That value is called a **feature** and is denoted as $x^{(j)}$.
- The **label** y_i can be either an element belonging to a finite set of **classes** $1, 2, ..., C$, or a real number, or a more complex structure, like a vector, a matrix, a tree or a graph. Unless otherwise stated, y_i is either one of a finite set of classes or a real number.

The goal of a **supervised learning algorithm** is to use the dataset to produce a **model** that takes a feature vector $\mathbf{x}$ as input and outputs information that allows deducing the label for this feature vector.

1.2.2 Unsupervised Learning

In **unsupervised learning**, the dataset is a collection of **unlabeled examples** $\{\mathbf{x}_i\}_{i=1}^N$. The goal of an **unsupervised learning algorithm** is to create a model that takes a feature vector $\mathbf{x}$ as input and

either transforms it into another vector or into a value that can be used to solve a practical problem. For example,

- in **clustering**, the model returns the id of the cluster for each feature vector in the dataset.
- in **dimensionality reduction**, the output of the model is a feature vector that has fewer features than the input $\mathbf{x}$;
- in **outlier detection**, the output is a real number that indicates how $\mathbf{x}$ is different from a "typical" example in the dataset.

1.2.3 Semi-Supervised Learning

In **semi-supervised learning**, the dataset contains both labeled and unlabeled examples.

- Usually the quantity of unlabeled examples is much higher than the number of labeled examples.

The goal of **semi-supervised learning algorithm** is the same as the goal of the supervised learning algorithm.

- The hope here is that using many unlabeled examples can help the learning algorithm to find a better model.

How does the learning benefit from adding more unlabeled examples?

- By adding unlabeled examples, you add more information about your problem: *a larger sample reflects better the probability distribution the data we labeled came from.*

1.2.4 Reinforcement Learning

Reinforcement learning is a subfield of machine learning where the machine "lives" in an environment and is capable of perceiving the *state* of that environment as vector of features.

- The machine can execute *actions* in every state. Different actions bring different *rewards* and could also move the machine to another state of the environment.

The goal of a reinforcement learning algorithm is to learn a *policy*. A policy is a function (similar to the model in supervised learning) that takes the feature vector of a state as input and outputs an optimal action to execute in that state. The action is optimal if it maximizes the *expected average reward*.

1.3 How Supervised Learning Works

Supervised learning is the type of machine learning most frequently used in practice. The supervised learning process starts with gathering the data. The data for supervised learning is a collection of pairs (input,output).

- Inputs could be anything, for example, email messages, pictures, or sensor measurements.
- Outputs are usually real numbers, or labels (e.g. "spam", "not_spam", etc). In some cases, outputs are vectors (e.g., four coordinates of the rectangle around a person on the picture), sequences (e.g. ["adjective", "adjective", "noun"] for the input "big beautiful car") or have some other structure.

You want to solve spam detection using supervised learning. First, you gather the data of 10,000 email messages and you have your label "spam" or "not_spam" for each message. With the data at hand, you have to represent each message using a feature vector.

A common approach it to use **bag of words**, is to take a dictionary of English words (e.g. 20,000 alphabetically sorted words) and stipulate that in our feature vector:

- the first feature is equal to 1 if the email message contains the word "a"; other wise, this feature is 0;
- the second feature is equal to 1 if the email message contains the word "aaron"; otherwise, this feature equals 0;
- ...
- the feature at position 20,000 is equal to 1 if the email message contains the word "zulu"; otherwise, this feature is equal to 0.

You repeat the above procedure for every email message in our collection, which gives us 10,000 feature vectors (each vector having the dimensionality of 20,000) and a label ("spam"/"not_spam").

We successfully converted input data into machine-readable type, but the output labels are still in the form of human-readable text. Some learning algorithms require transforming labels into numbers. For demonstration purpose, we will use a supervised learning algorithm called **Support Vector Machine** (SVM). This algorithm requires the positive label (in our case it's "spam") has the numeric value of +1 (one), and the negative label ("not_spam") has the value of -1 (minus one).

You now have a **dataset** and a **learning algorithm**, you will need to apply the learning algorithm to the dataset to get the **model**.

SVM sees every feature vector as a point in a high-dimensional space (in our case, space is 20,000 – dimensional). The algorithm puts all feature vectors on an imaginary 20,000-dimensional plot and draws an imaginary 19,999 – dimensional line (a *hyperplane*) that separates examples with positive labels from examples with negative labels. In machine learning, the boundary separating the examples of different classes is called the **decision boundary**.

Hyperplane is denoted by two **parameters**

$$\mathbf{wx} - b = 0,$$

- a real-valued vector $\mathbf{w}$ of the same dimensionality as our input vector $\mathbf{x}$, and a real number b
- $\mathbf{wx} = w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \ldots + w^{(D)}x^{(D)}$, and D is the number of dimensions of the feature vector $\mathbf{x}$.

The predicted label for some input feature vector $\mathbf{x}$ is given like this:

$$y = \text{sign}(\mathbf{w}\mathbf{x} - b),$$

where `sign` is a mathematical operator that takes any value as input and returns $+1$ if the input is a positive number or -1 if the input is a negative number.

The goal of the learning algorithm, SVM in this case, is to leverage the dataset and find the optimal values $\mathbf{w}^*$ and b^* for parameters $\mathbf{w}$ and b. Once the learning algorithm identifies these optimal values, the **model** $f(\mathbf{x})$ is then defined as:

$$f(\mathbf{x}) = \text{sign}\left(\mathbf{w}^*\mathbf{x} - b^*\right)$$

How do we find these optima values? It turns out it is an optimization problem. Machines are good at optimizing functions under constraints.

$$\begin{aligned} \mathbf{w}\mathbf{x}_i - b \geq +1 \quad & \text{if } y_i = +1 \\ \mathbf{w}\mathbf{x}_i - b \leq -1 \quad & \text{if } y_i = -1 \end{aligned}$$

It will also be ideal if the hyperplane separates positive examples from negative ones with the largest **margin**. The margin is the distance between the closestt examples of two classes, as defined by the decision boundary. A large margin contributes to a better **generalization**, that is how well the model will classify new examples in the future.

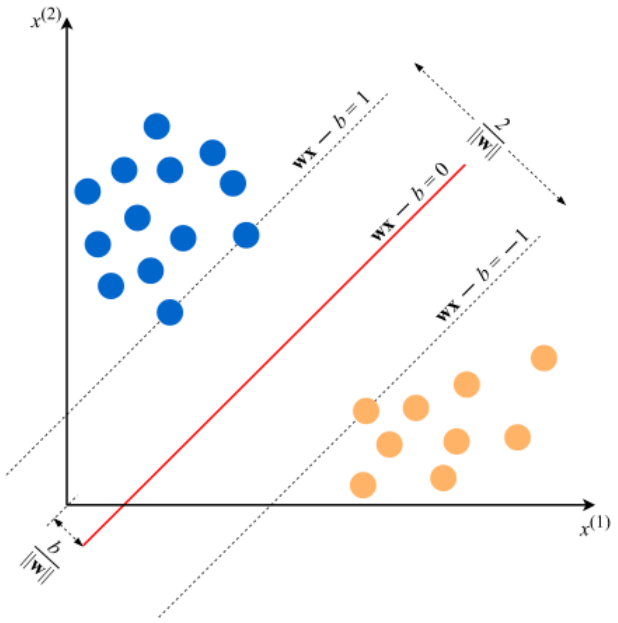


Figure 1.1: An example of an SVM model for two-dimensional feature vectors.

Let's do some quick refreshment

Distance Formulas in Euclidean Space

Distance Between Two Points
In two-dimensional space, for points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$, the distance is calculated as:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Distance From a Point to a Line
In two-dimensional space, for a line defined by $ax + by + c = 0$ and a point $P(x_0, y_0)$, the distance is:

$$d = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

Distance Between Two Parallel Lines
For two parallel lines with equations $ax + by + c_1 = 0$ and $ax + by + c_2 = 0$, the distance is:

$$D = \frac{|c_2 - c_1|}{\sqrt{a^2 + b^2}}$$

So, the optimization problem that we want the machine to solve looks like this:

Minimize $\|\mathbf{w}\|$ *subject to* $y_i(\mathbf{w}\mathbf{x}_i - b) \geq 1$ *for* $i = 1, \ldots, N$. *The expression* $y_i(\mathbf{w}\mathbf{x}_i - b) \geq 1$ *is just a compact way to write the above two constraints.*

More on SVMs later. This simple example should give you an idea how supervised learning works.

1.4 Why the Model Works on New Data

Let's refer to Figure 1.1. If two classes are separable from one another by a decision boundary, then, obviously, examples that belong to each class are located in two different subspaces which the decision boundary creates.

If the examples used for training were selected randomly, independently of one another, and following the same procedure, then statistically, it is *more likely* that the new negative example will be located on the plot somewhere not too far from other negative examples. The idea goes with the positive examples as well.

Chapter 2

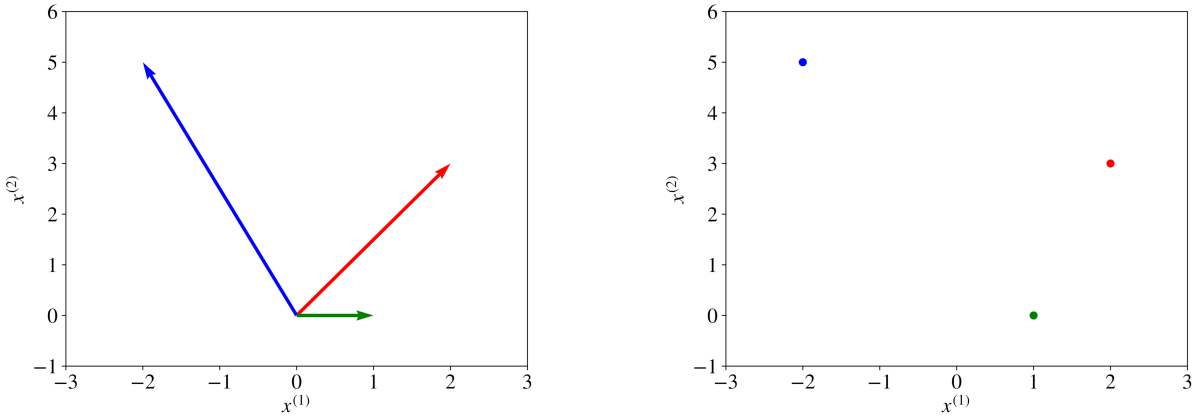
Notation and Definition

2.1 Notation

We will review all the necessary notation and mathematics for us to continue the journey of Machine Learning.

2.1.1 Data Structure

A **scalar** is simple numerical value, like 15 or -3.25, denoted by an italic letter, like x or a. A **vector** is an ordered list of scalar values, called attributes. Vector is denoted by bold character $\mathbf{x}$ or $\mathbf{w}$. A **matrix** is a rectangular array of numbers arranged in rows and columns.



(a) First subfigure

(b) Second subfigure

Figure 2.1: Three vectors visualized as directions and as points.

A **matrix** is a rectangular array of numbers arranged in rows and columns, which are denoted with bold capital letters, such as $\mathbf{A}$ or $\mathbf{W}$. A **set** is an unordered collection of unique elements. We denote a set as a calligraphic capital character, for example, $\mathcal{S}$. When an element belongs to a set $\mathcal{S}$, we write $x \in \mathcal{S}$. We can obtain a new set $\mathcal{S}_3$ as an **intersection** of two set $\mathcal{S}_1$ and $\mathcal{S}_2$, written as $\mathcal{S} \leftarrow \mathcal{S}_1 \cap \mathcal{S}_2$. Also we can obtain a new set by **union**, $\mathcal{S}_3 \leftarrow \mathcal{S}_1 \cup \mathcal{S}_2$.

2.1.2 Capital Sigma Notation

The summation over a collection $\mathcal{X} = \{x_1, x_2, \ldots, x_{n-1}, x_n\}$ or over the attributes of a vector $\mathbf{x} = \left[x^{(1)}, x^{(2)}, \ldots, x^{(m-1)}, x^{(m)}\right]$ is denoted like this:

$$\sum_{i=1}^{n} x_i \overset{\text{def}}{=} x_1 + x_2 + \ldots + x_{n-1} + x_n, \text{ or else: } \sum_{j=1}^{m} x^{(j)} \overset{\text{def}}{=} x^{(1)} + x^{(2)} + \ldots + x^{(m-1)} + x^{(m)}$$

The notation $\stackrel{\text{def}}{=}$ means "is defined as".

2.1.3 Capital Pi Notation

$$\prod_{i=1}^{n} x_i \stackrel{\text{def}}{=} x_1 \cdot x_2 \cdot \ldots \cdot x_{n-1} \cdot x_n$$

- A product of elements in a collection or attributes of a vector.

2.1.4 Operations on Sets

Given the expression:

$$\mathcal{S}' \leftarrow \left\{x^2 \mid x \in \mathcal{S}, x > 3\right\}$$

This notation is used to define a derived set creation operator. It means that we create a new set $\mathcal{S}'$ by including the square of each element x from the set $\mathcal{S}$, under the condition that x is greater than 3. In other words, $\mathcal{S}'$ is comprised of the squares of all elements in $\mathcal{S}$ which are greater than 3.

Additionally, the cardinality operator $|\mathcal{S}|$ is used to denote the number of elements in the set $\mathcal{S}$. For example, if $\mathcal{S} = \{1, 2, 4, 5\}$, then $\mathcal{S}' = \{16, 25\}$ as only 4 and 5 from $\mathcal{S}$ satisfy the condition $x > 3$. The **cardinality** $|\mathcal{S}|$ in this case would be 4.

2.1.5 Operations on Vectors

Vector Addition and Subtraction: The sum and difference of two vectors $\mathbf{x}$ and $\mathbf{z}$ are defined component-wise as:

$$\mathbf{x} + \mathbf{z} = \left[x^{(1)} + z^{(1)}, \ldots, x^{(m)} + z^{(m)}\right]$$

$$\mathbf{x} - \mathbf{z} = \left[x^{(1)} - z^{(1)}, \ldots, x^{(m)} - z^{(m)}\right]$$

Example: For $\mathbf{x} = [1, 2]$ and $\mathbf{z} = [3, 4]$,

$$\mathbf{x} + \mathbf{z} = [1 + 3, 2 + 4] = [4, 6]$$

Scalar Multiplication: A vector multiplied by a scalar c results in a scaled vector:

$$\mathbf{x}c = \left[cx^{(1)}, \ldots, cx^{(m)}\right]$$

Example: For $\mathbf{x} = [1, 2]$ and $c = 3$,

$$\mathbf{x}c = [3 \times 1, 3 \times 2] = [3, 6]$$

Dot Product: The dot product of two vectors $\mathbf{w}$ and $\mathbf{x}$ is a scalar:

$$\mathbf{w}\mathbf{x} = \sum_{i=1}^{m} w^{(i)} x^{(i)}$$

Example: For $\mathbf{w} = [1, 2]$ and $\mathbf{x} = [3, 4]$,

$$\mathbf{w}\mathbf{x} = 1 \times 3 + 2 \times 4 = 3 + 8 = 11$$

Matrix-Vector Multiplication: Multiplying a matrix $\mathbf{W}$ by a vector $\mathbf{x}$ yields another vector. For example:

$$\begin{aligned}\mathbf{W}\mathbf{x} &= \begin{bmatrix} w^{(1,1)} & w^{(1,2)} & w^{(1,3)} \\ w^{(2,1)} & w^{(2,2)} & w^{(2,3)} \end{bmatrix} \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \end{bmatrix} \\ &\stackrel{\text{def}}{=} \begin{bmatrix} w^{(1,1)}x^{(1)} + w^{(1,2)}x^{(2)} + w^{(1,3)}x^{(3)} \\ w^{(2,1)}x^{(1)} + w^{(2,2)}x^{(2)} + w^{(2,3)}x^{(3)} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{w}^{(1)}\mathbf{x} \\ \mathbf{w}^{(2)}\mathbf{x} \end{bmatrix}\end{aligned}$$

Example: For

$$\mathbf{W} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \text{ and } \mathbf{x} = \begin{bmatrix} 5 \\ 6 \end{bmatrix},$$

$$\mathbf{W}\mathbf{x} = \begin{bmatrix} 1 \times 5 + 2 \times 6 \\ 3 \times 5 + 4 \times 6 \end{bmatrix} = \begin{bmatrix} 17 \\ 39 \end{bmatrix}$$

Transpose and Multiplication: For the transpose of a vector $\mathbf{x}$ denoted $\mathbf{x}^\top$, and a matrix $\mathbf{W}$, the multiplication $\mathbf{x}^\top\mathbf{W}$ is given by:

$$\begin{aligned}\mathbf{x}^\top\mathbf{W} &= \begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix} \begin{bmatrix} w^{(1,1)} & w^{(1,2)} & w^{(1,3)} \\ w^{(2,1)} & w^{(2,2)} & w^{(2,3)} \end{bmatrix} \\ &\stackrel{\text{def}}{=} \left[w^{(1,1)}x^{(1)} + w^{(2,1)}x^{(2)}, w^{(1,2)}x^{(1)} + w^{(2,2)}x^{(2)}, w^{(1,3)}x^{(1)} + w^{(2,3)}x^{(2)}\right]\end{aligned}$$

Example: For

$$\mathbf{x} = \begin{bmatrix} 7 \\ 8 \end{bmatrix} \text{ and } \mathbf{W} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix},$$

$$\mathbf{x}^\top\mathbf{W} = \begin{bmatrix} 7 \times 1 + 8 \times 4, 7 \times 2 + 8 \times 5, 7 \times 3 + 8 \times 6 \end{bmatrix} = \begin{bmatrix} 39, 54, 69 \end{bmatrix}$$

2.1.6 Functions

Definition of a Function
A function is a relation that associates each element x of a set $\mathcal{X}$, known as the domain, to a single element y of another set $\mathcal{Y}$, known as the codomain. This relation is denoted as $y = f(x)$, where f is the name of the function, x is the input or argument, and y is the output. The input variable is also referred to as the variable of the function.

Example: Consider the function $f(x) = x^2$ defined on the domain $\mathcal{X} = \mathbb{R}$. For $x = 2$, the output is $f(2) = 2^2 = 4$.

Local and Global Minima
The function $f(x)$ has a local minimum at $x = c$ if $f(x) \geq f(c)$ for every x in an open interval around c. An open interval, such as $(0, 1)$, includes all numbers between its endpoints but not the endpoints themselves. The smallest value among all local minima is known as the global minimum.

Example: In the function $f(x) = (x - 1)^2$, the local (and global) minimum occurs at $x = 1$ since $f(x) \geq f(1) = 0$ for all x.

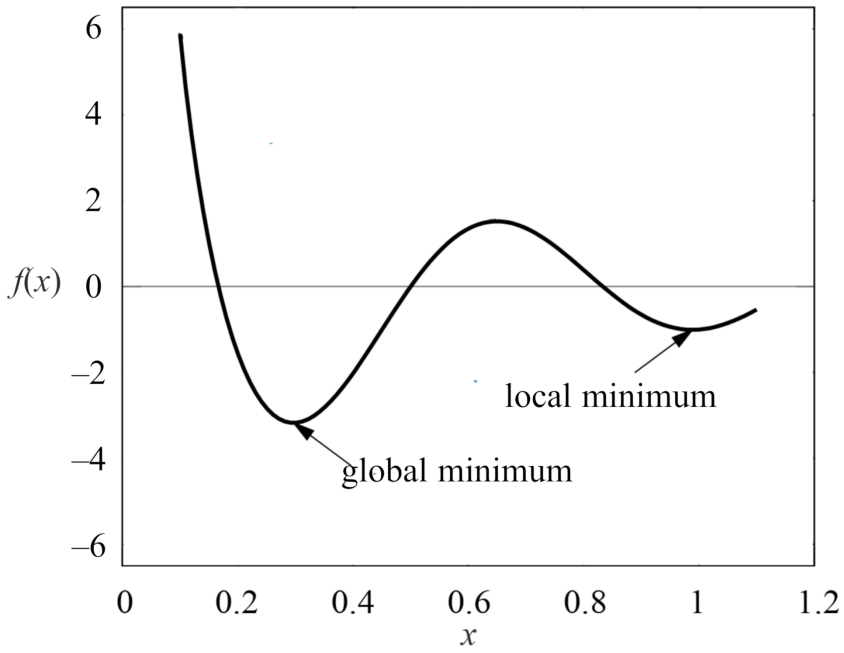


Figure 2.2: A local and a global minima of a function.

Vector Functions
A vector function, denoted $\mathbf{y} = \mathbf{f}(x)$, is a function that returns a vector $\mathbf{y}$. Its argument can be either a vector or a scalar.

Example: For the vector function $\mathbf{f}(x) = [x, x^2]$, with $x = 2$, the output is $\mathbf{f}(2) = [2, 2^2] = [2, 4]$.

2.1.7 Max and Arg Max

Given a set of values $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$, the operator $\max_{a \in \mathcal{A}} f(a)$ returns the highest value of $f(a)$ for all elements in the set $\mathcal{A}$. Conversely, the operator $\arg\max_{a \in \mathcal{A}} f(a)$ identifies the specific element a in the set $\mathcal{A}$ that maximizes the function $f(a)$.

In cases where the set is implicit or infinite, we can use the notation $\max_a f(a)$ or $\arg\max_a f(a)$ respectively. Similarly, the operators min and arg min function in a comparable way, determining the lowest value of a function and the

2.1.8 Assignment Operator

The expression $a \leftarrow f(x)$ means that the variable a gets the new value: the result of $f(x)$. We say that the variable a gets assigned a new value. Similarly, $\mathbf{a} \leftarrow [a_1, a_2]$ means that the vector variable $\mathbf{a}$ gets the two-dimensional vector $[a_1, a_2]$.

2.1.9 Derivative and Gradient

A **derivative** f' of a function f is a function or a value that describes how fast f grows (or decreases). If the derivative f' is a function, then the function f can grow at a different pace in different regions of its domain.

we can use **chain rule** when we encounter hard-to-differentiate function. For instance if $F(x) = f(g(x))$, where f and g are some functions, then $F'(x) = f'(g(x))g'(x)$.

Gradient is the generalization of derivative for functions that take several inputs (or one input in the form of a vector or some other complex structure). A gradient of a function is a vector of **partial derivatives**. For example, $f\left(\left[x^{(1)}, x^{(2)}\right]\right) = ax^{(1)} + bx^{(2)} + c$, then the partial derivative of function f *with respect to* $x^{(1)}$, denoted as $\frac{\partial f}{\partial x^{(1)}}$, is given by,

$$\frac{\partial f}{\partial x^{(1)}} = a + 0 + 0 = a$$

where a is the derivative of the function $ax^{(1)}$; the two zeroes are respectively derivatives of $bx^{(2)}$ and c, because $x^{(2)}$ is considered constant when we compute the derivative with respect to $x^{(1)}$, and the derivative of any constant is zero. Similarly, the partial derivative of function f with respect to $x^{(2)}$, $\frac{\partial f}{\partial x^{(2)}}$, is given by,

$$\frac{\partial f}{\partial x^{(2)}} = 0 + b + 0 = b$$

The gradient of function f, denoted as ∇f is given by the vector $\left[\frac{\partial f}{\partial x^{(1)}}, \frac{\partial f}{\partial x^{(2)}}\right]$.

2.2 Random Variable

A **random variable**, usually written as an italic letter, like X, is a variable whose possible values are numerical outcomes of a random phenomenon. There are two types of random variables: **discrete** and **continuous**. A **discrete random variable** takes on only countable number of distinct values such as *red*, *yellow*, *blue* or 1,2,3, ...

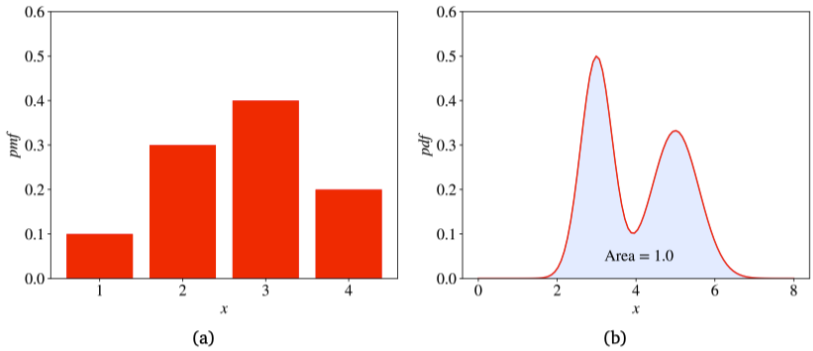


Figure 2.3: A probability mass function and a probability density function.

The **probability distribution** of a discrete random variable is described by a list of probability asso-

ciated with each of its possible values. This list of probability is called a **probability mass function** (pmf)(Fig.2.3, **a**).

A **continuous random variable** takes an infinite number of possible values in some interval. The probability distribution of a continual random variable (a continuous probability distribution) is described by a **probability density function** (pdf) (Fig.2.3, **b**).

Let a discrete random variable X have k possible values $\{x_i\}_{i=1}^{k}$. The **expectation** of X denoted as $\mathbb{E}[X]$ is given by,

$$\begin{aligned}\mathbb{E}[X] &\stackrel{\text{def}}{=} \sum_{i=1}^{k} [x_i \cdot \Pr(X = x_i)] \\ &= x_1 \cdot \Pr(X = x_1) + x_2 \cdot \Pr(X = x_2) + \cdots + x_k \cdot \Pr(X = x_k)\end{aligned}$$

where $\Pr(X = x_i)$ is the probability that X has the value x_i according to the pmf. The expectation of a random variable is also called the **mean**, **average** or **expected value** and is frequently denoted with the letter μ,

Now the **standard deviation**, defined as,

$$\sigma \stackrel{\text{def}}{=} \sqrt{\mathbb{E}[(X - \mu)^2]}$$

Part II

Advance Machine Learning

Part III

Convolution Neural Networks

Part IV

Adversarial Attacks and Training

Chapter 3

Intriguing Properties of Neural Network

Deep neural networks, known for their exceptional performance in speech and visual recognition tasks, exhibit two notable characteristics (Szegedy et al., 2013). First, *the semantic information in their higher layers is embedded not in individual units but in the collective space they form.* This insights shifts the focus from analyzing single neurons to considering the entire unit group to understand network processing. Second, *these networks display a surprisingly sensitivity to minute, yet percisely tailored alternations (or perturbation).* Such small changes can lead to incorrect outcomes. This vulnerability is not due to random noise; the same modifications can deceive different networks trained on a different subset of the dataset, to misclassify the same input.

3.1 Introduction

Deep neural networks are powerful learning models that achieve excellent performance on visual and speech recognition problems because they can express arbitrary computation that consists of a modest number of massively parallel nonlinear steps. As the resulting computation is automatically discovered by backpropagation via supervised learning, it can be difficult to interpret and can have counter-intuitive properties.

The **first** property is concerned with the semantic meaning of individual units. It seems that the entire space of activation, rather than the individual units, that contains the bulk of the semantic information contrary to prior belief and the **second** property is concerned with the stability of neural networks with respect to small perturbation to their inputs. Apply an *imperceptible* non-random perturbation to a test image, it is possible to arbitrarily change the network's prediction. These perturbation are found by optimizing the input to maximize the prediction error. The perturbed examples are often called "adversarial examples"

3.2 Framework

Notation $x \in \mathbb{R}$ denotes an input image, $\phi(x)$ is an activation values of some layer. (Szegedy et al., 2013) first examine properties of the image of $\phi(x)$, and then search for its blind spots.

Part V

Recurrent Neural Networks

Part VI

Transformers

Part VII

Artificial General Intelligence

Chapter 4

Continual Learning

4.1 Introduction

Continual Learning is motivated by the fact that human and other organisms has the ability to adapt, accumulate and exploit knowledge. A common setting for continual learning is to learn a sequence of contents one by one and behave as if they were observed simultaneously (Wang et al., 2023). Each task learned throughout the life time can be new skills, new examples of old skills, different environments, etc (Fig.4.1, a). This attribute of continual learning makes it also referred to as **incremental learning** or **lifelong learning**.

Unlike conventional pipeline, where joint training is applied, continual learning is characterized by learning from dynamic data distributions. A major challenge is known as **catastrophic forgetting**, where *adaptation to a new distribution generally results in a largely reduced ability to capture the old ones*. This dilemma is a facet of the trade-off between **learning plasticity** and **memory stability**: an excess of the former interferes with the latter, and vice versa. A good continual learning algorithm should obtain a strong **generalizability** to accommodate distribution differences within and between tasks (Fig.4.1, b). As a naive baseline, retraining all old training samples (if allowed) makes it easy to address the above challenges, but creates huge computational and storage overheads (as well as potential privacy issues). In fact, continual learning is primarily intended to ensure **resource efficiency** of model updates, preferably close to learning only new training samples.

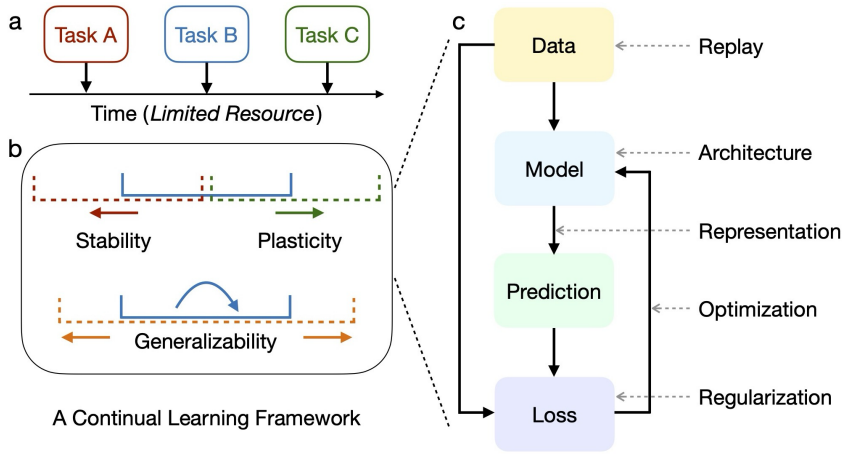


Figure 4.1: A conceptual framework of continual learning. **a**, Continual learning requires adapting to incremental tasks with dynamic data distributions. **b**, A desirable solution should ensure a proper balance between stability (red arrow) and plasticity (green arrow), as well as an adequate generalizability to intra-task (blue arrow) and inter-task (orange arrow) distribution differences. **c**, Representative strategies have targeted various aspects of machine learning.

Numerous efforts have been devoted to addressing the above challenges, which can be conceptually separated into five groups (Fig.4.1, c): *regularization-based approach*; *replay-based approach*; *optimization-based approach*; *representation-based approach*; and *architecture-based approach*. These methods are *closely connected*, e.g., regularization and replay ultimately act to rectify the gradient directions, and *highly synergistic*, e.g., the efficacy of replay can be facilitated by distilling knowledge from the old model.

4.2 Setup

In this section, we first present a basic formulation of continual leanring. Then we introduce typical scenairos and evaluation metrics.

4.2.1 Basic Formulation

A continual learning model parameterized by θ needs to learn corresponding task(s) with no or limited access to old training samples and perform well on their test sets. Formally, an incoming batch of training samples belonging to a task t can be represented as $\mathcal{D}_{t,b} = \left\{\mathcal{X}_{t,b}, \mathcal{Y}_{t,b}\right\}$ where $\mathcal{X}_{t,b}$ is the input data, $\mathcal{Y}_{t,b}$ is the data label, $t \in \mathcal{T} = \{1, \cdots, k\}$ is the task identity and $b \in \mathcal{B}_t$ is the batch index ($\mathcal{T}$ and $\mathcal{B}_t$ denote their space, respectively). Here we define a "task" by its training samples $\mathcal{D}_t$ following the distribution $\mathbb{D}_t := p\left(\mathcal{X}_t, \mathcal{Y}_t\right)$ ($\mathcal{D}_t$ denotes the entire training set by omitting the batch index, likewise for $\mathcal{X}_t$ and $(\mathcal{Y}_t)$, and assume that there is no difference in distribution between training and testing. Under realistic constraints, the data label $\mathcal{Y}_t$ and the task identity t might not be always available. In continual learning, the training samples of each task can arrive incrementally in batches (i.e., $\left\{\left\{\mathcal{D}_{t,b}\right\}_{b\in\mathcal{B}_t}\right\}_{t\in\mathcal{T}}$) or simultaneously (i.e.,$\{\mathcal{D}_t\}_{t\in\mathcal{T}}$).

Scenario	Training	Testing
IIL	$\left\{\left\{\mathcal{D}_{t,b}, t\right\}_{b\in\mathcal{B}_t}\right\}_{t=j}$	$\{p\left(\mathcal{X}_t\right)\}_{t=j;t}$ is not required
DIL	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}; p\left(\mathcal{X}_i\right) \neq p\left(\mathcal{X}_j\right)$ and $\mathcal{Y}_i = \mathcal{Y}_j$ for $i \neq j$	$\{p\left(\mathcal{X}_t\right)\}_{t\in\mathcal{T}}, t$ is not required
TIL	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}; p\left(\mathcal{X}_i\right) \neq p\left(\mathcal{X}_j\right)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p\left(\mathcal{X}_t\right)\}_{t\in\mathcal{T};t}$ is available
CIL	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}; p\left(\mathcal{X}_i\right) \neq p\left(\mathcal{X}_j\right)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p\left(\mathcal{X}_t\right)\}_{t\in\mathcal{T};t}$ is unavailable
TFCL	$\left\{\left\{\mathcal{D}_{t,b}\right\}_{b\in\mathcal{B}_t}\right\}_{t\in\mathcal{T}}; p\left(\mathcal{X}_i\right) \neq p\left(\mathcal{X}_j\right)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p\left(\mathcal{X}_t\right)\}_{t\in\mathcal{T}}; t$ is optionally available
OCL	$\left\{\left\{\mathcal{D}_{t,b}\right\}_{b\in\mathcal{B}_t}\right\}_{t\in\mathcal{T}}, \lvert b\rvert = 1; p\left(\mathcal{X}_i\right) \neq p\left(\mathcal{X}_j\right)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p\left(\mathcal{X}_t\right)\}_{t\in\mathcal{T}}; t$ is optionally available
BBCL	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}; p\left(\mathcal{X}_i\right) \neq p\left(\mathcal{X}_j\right), \mathcal{Y}_i \neq \mathcal{Y}_j$ and $\mathcal{Y}_i \cap \mathcal{Y}_j \neq \emptyset$ for $i \neq j$	$\{p\left(\mathcal{X}_t\right)\}_{t\in\mathcal{T};t}$ is unavailable
CPT	$\left\{\mathcal{D}_t^{pt}, t\right\}_{t\in\mathcal{T}_{pt}}$, followed by a downstream task j	$\{p\left(\mathcal{X}_t\right)\}_{t=j;t}$ is not required

Table 4.1: A formal comparison of typical continual learning scenarios. $\mathcal{D}_{t,b}$: the training samples of task t and batch b. $|b|$: the size of batch b. $\mathcal{B}_t$: the space of incremental batches belonging to task t. $\mathcal{D}_t$: the training set of task t (further specified as $\mathcal{D}_t^{pt}$ for pre-training). $\mathcal{T}$: the space of all incremental tasks (further specified as $\mathcal{T}^{pt}$ for pre-training). $\mathcal{X}_t$: the input data in $\mathcal{D}_t$, $p\left(\mathcal{X}_t\right)$: the distribution of $\mathcal{X}_t$.$\mathcal{Y}_t$: the data label of $\mathcal{X}_t$.

4.2.2 Typical Scenairo

Detail of some tyical continual learning scenairos (refer to Table 4.1 for a formal comparision):

- *Instance-Incremental Learning* (IIL): All training samples belong to the same task and arrive in batches.
- *Domain-Incremental Learning* (DIL): Tasks have the same data label space but different input distributions. Task identities are not required.

- ***Task-Incremental Learning*** (TIL): Tasks have disjoint data label spaces. Task identities are provided in both training and testing.
- ***Class-Incremental Learning*** (CIL): Tasks have disjoint data label spaces. Task identities are only provided in training.
- *Task-free Continual Learning* (TFCL): Tasks have disjoint data label spaces. Task identities are not provided in either training or testing.
- *Online Continual Learning* (OCL): Tasks have disjoint data label spaces. Training samples for each task arrive as a one-pass data stream.
- *Blurred Boundary Continual Learning* (BBCL): Task boundaries are blurred, characterized by distinct but overlapping data label spaces.
- *Continual Pre-training* (CPT): Pre-training data arrives in sequence. The goal is to improve the performance of learning downstream tasks.

Lots of the above mentioned scenairo is messy, hence we will focus on the most popular scenairos: Task-Incremental Learning and Class-Incremental Learning.

4.2.3 Evaluation Metrics

Overall performance is typically evaluated by *average accuracy* (AA) and *average incremental accuracy* (AIA). Let $a_{k,j} \in [0,1]$ denote the classification accuracy evaluated on the test set of the j-th task after incremental learning of the k-th task ($j \leq k$). The output space to compute $a_{k,j}$ consists of the classes in either $\mathcal{Y}_j$ or $\cup_{i=1}^{k}\mathcal{Y}_i$, corresponding to the use of multi-head evaluation (e.g., TIL) or single-head evaluation (e.g., CIL). The two metrics at the k-th task are then defined as

$$\mathrm{AA}_k = \frac{1}{k}\sum_{j=1}^{k} a_{k,j}$$

AA represnets the overall performance at the current moment.

$$\mathrm{AIA}_k = \frac{1}{k}\sum_{i=1}^{k} \mathrm{AA}_i$$

AIA reflects the historical variaion.

Memory stability can be evaluted by *forgetting measure* (FM) and *backward transfer* (BWT). As for the former, the forgetting of a task is calculated by the difference between its maximum performance obtained in the past and its current performance:

$$f_{j,k} = \max_{i\in\{1,\ldots,k-1\}} \left(a_{i,j} - a_{k,j}\right), \forall j < k$$

Bibliography

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *arXiv preprint arXiv:2302.00487*, 2023.