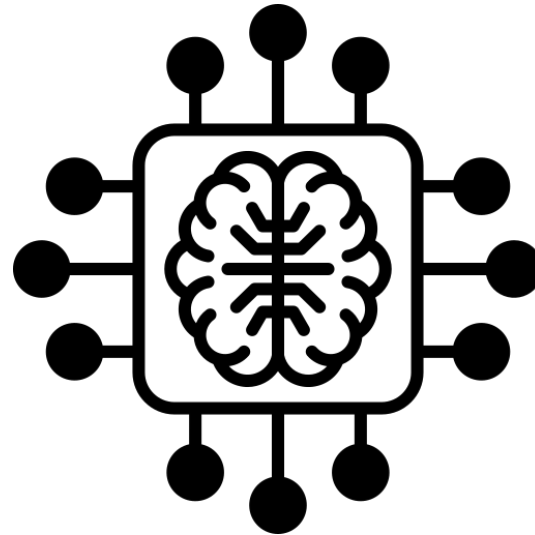


SBS4115 Fundamentals of AI & Data Analytics



Neural Networks and Deep Learning

Lecturer: Ir Dr Kelvin K. W. Siu

email: kelvinsiu@thei.edu.hk



Department of Construction,
Environment and Engineering

Intended Learning Outcomes

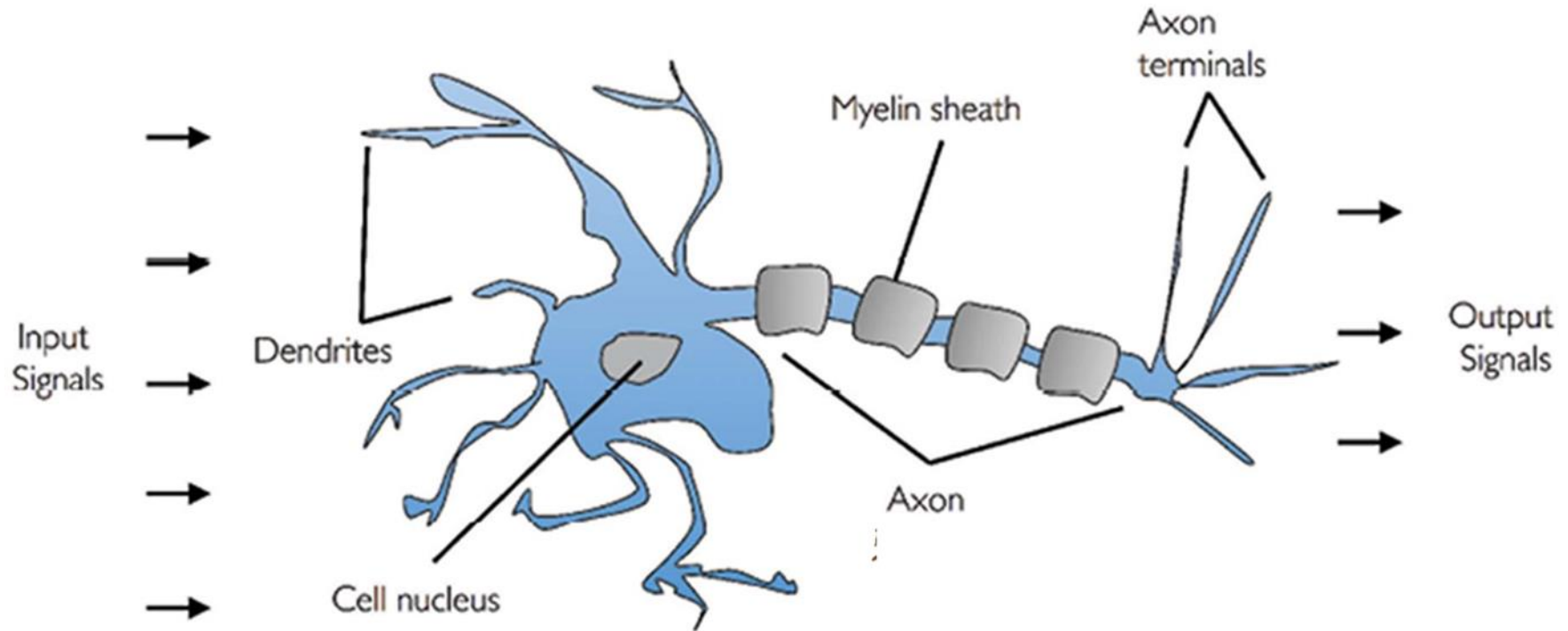


- By the end of this lecture, you will be able to...
 - Describe how artificial neurons work and what the perceptron learning rule is
 - Discriminate all data points into two linearly separable classes
 - Discuss the three possible cases of perceptron algorithm
 - Introduce Python data analysis (Pandas)
 - Apply descriptive statistics and arithmetic in Pandas

Artificial Neurons and Perceptron Learning Rule



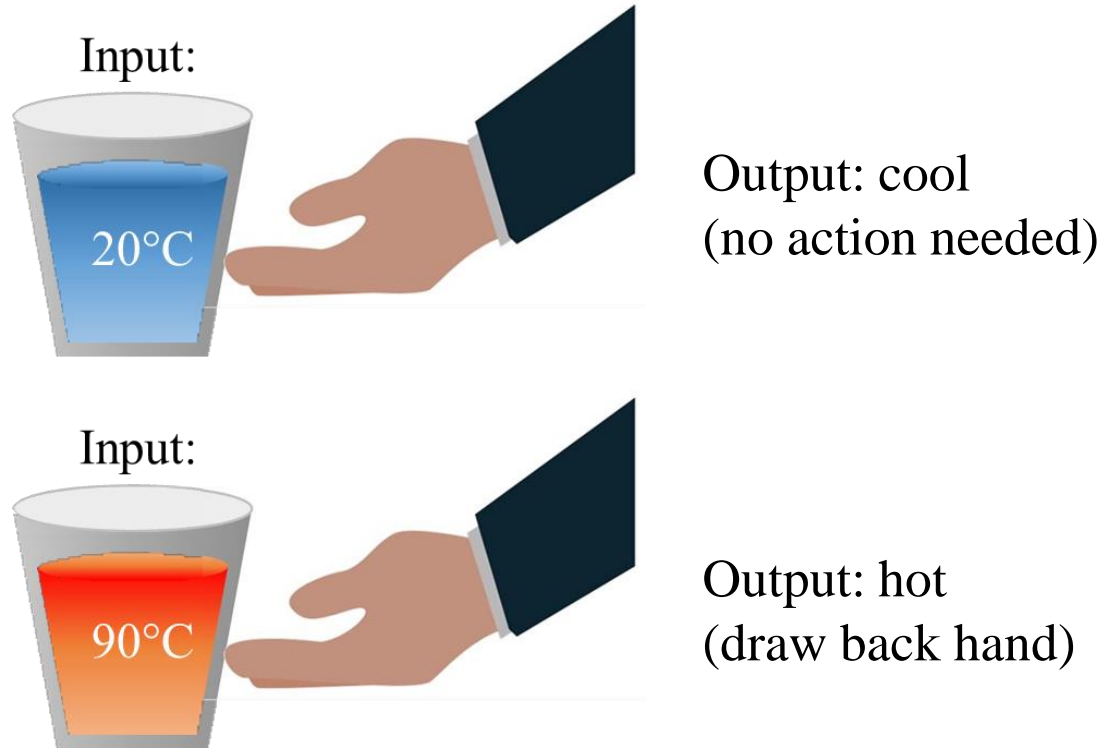
- In this chapter, we will discuss the perceptron and other machine learning algorithms.
- To start with, we need to review the beginning of machine learning.
- Trying to understand how the biological brain works in order to design artificial intelligence (AI), *Warren McCulloch* and *Walter Pitts* published the first concept of a simplified brain cell, **the McCulloch-Pitts (MCP) neuron**, in 1943.
- Biological neurons are interconnected nerve cells in the brain that are involved in the processing and transmitting of chemical and electrical signals, which is illustrated in the figure on the next slide.



- *McCulloch* and *Pitts* described such a nerve cell as a simple logic gate with **binary outputs**; multiple signals arrive at the **dendrites**, they are then integrated into the cell body, and, if the accumulated signal **exceeds a certain threshold**, an **output signal** is generated that will be passed on by the axon.

Artificial Neurons and Perceptron Learning Rule

- How neuron works: a daily life example



But how did you learn if a cup of water is “cool” or “hot”?
What is the cutting line between “cool” and “hot”?

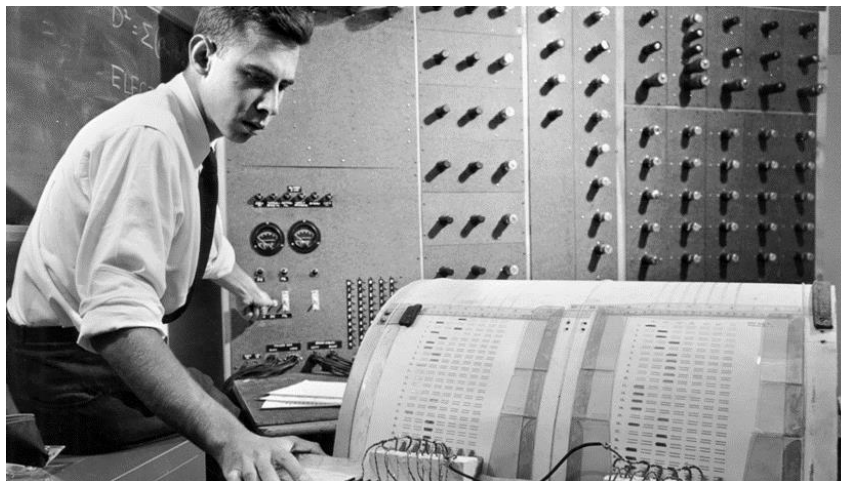
Artificial Neurons and Perceptron Learning Rule

- The learning process:



Artificial Neurons and Perceptron Learning Rule

- Only a few years later, *Frank Rosenblatt* published the first concept of the **perceptron learning rule** based on the MCP neuron model.
- With his perceptron rule, *Rosenblatt* proposed an algorithm that would **automatically learn the optimal weight coefficients** that would then be **multiplied with the input features** in order to make the decision of whether a neuron fires (transmits a signal) or not.



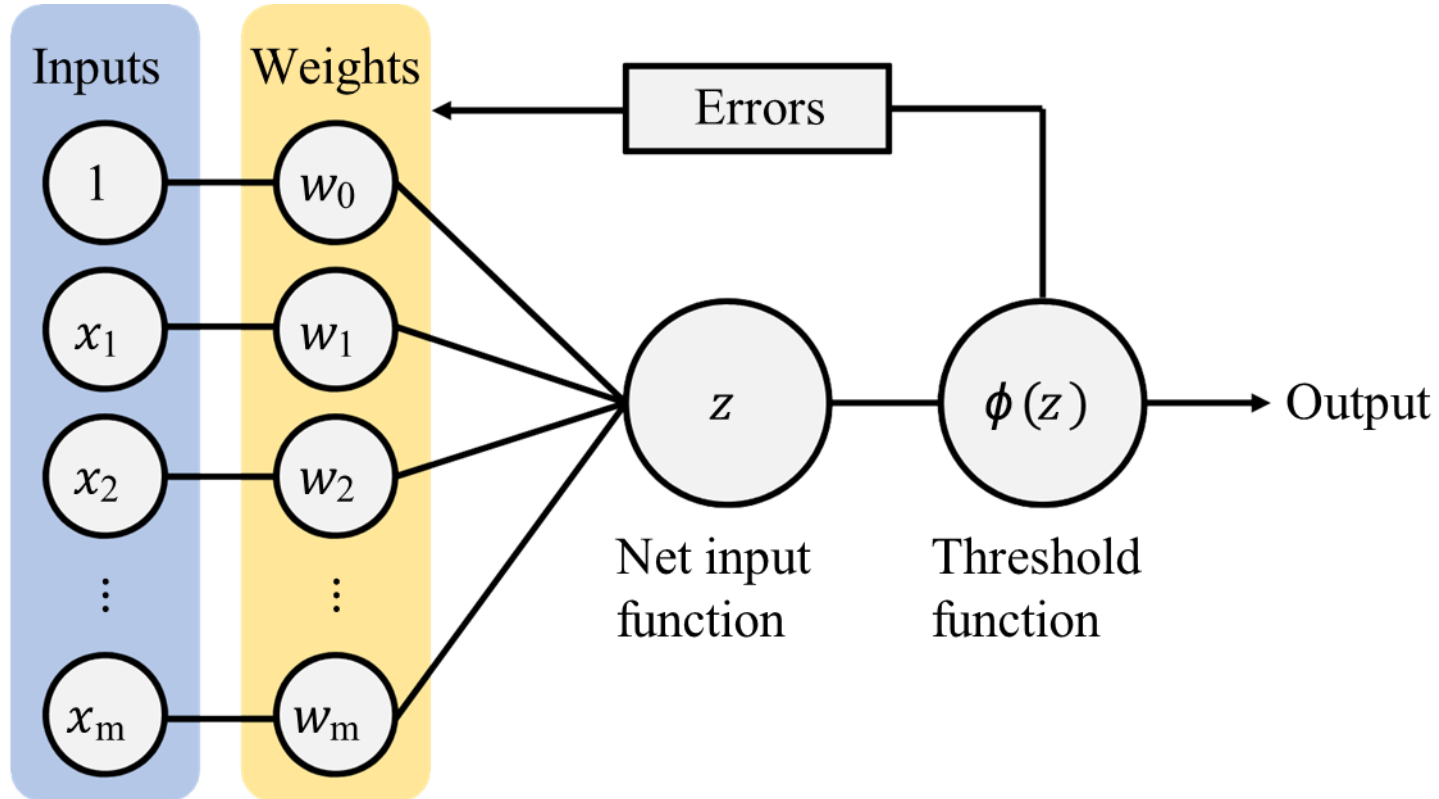
Artificial Neurons and Perceptron Learning Rule

- In the context of supervised learning and classification, such an algorithm could then be **used to predict whether a new data point belongs to one class or the other.**



Artificial Neurons and Perceptron Learning Rule

- Perceptron learning rule



Artificial Neurons and Perceptron Learning Rule

- More formally, we can put the idea behind artificial neurons into the context of a **binary classification task** where we refer to our two classes as 1 (positive class) and -1 (negative class) for simplicity.
- We can then define a **decision function $\phi(z)$** that takes a linear combination of certain input values \vec{x} and a corresponding weight vector \vec{w} :

$$\vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \quad \vec{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

where z is the net input defined by:

$$z = \vec{w} \cdot \vec{x} = w_1 x_1 + \cdots + w_m x_m$$

Artificial Neurons and Perceptron Learning Rule

- If the net input of a particular example, $\mathbf{x}^{(i)}$ is greater than a defined threshold θ , we predict class 1 and class -1 otherwise.
- In the perceptron algorithm, the decision function is a variant of a **unit step function**:

$$\phi(z) = \begin{cases} 1, & z \geq \theta \\ -1, & z < \theta \end{cases}$$

- For simplicity, we can also bring the θ to left side, let $w_0 = -\theta$, $x_0 = 1$, then:

$$z = w_0x_0 + w_1x_1 \dots + w_mx_m$$

$$\phi(z) = \begin{cases} 1, & z \geq 0 \\ -1, & z < 0 \end{cases}$$

- The term $w_0 = -\theta$ is called bias unit.

Artificial Neurons and Perceptron Learning Rule

- Let's consider a simple example with only two variables, x_1 and x_2 .
- Suppose after some computations we have the weights $w_0 = -4$, $w_1 = -1$, $w_2 = 2$, the net input becomes:

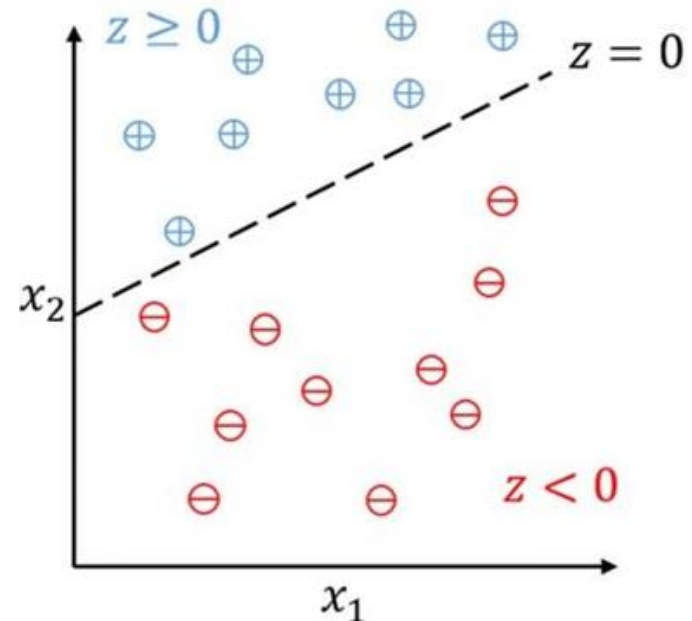
$$z = -4 - x_1 + 2x_2$$

- This gives the decision boundary:

$$-4 - x_1 + 2x_2 = 0$$

discriminating all data points into two
linearly separable classes:

i.e. positive class and negative class.



Artificial Neurons and Perceptron Learning Rule



Q: Now the problem is, how can we obtain suitable values of the weights, so that the decision line can accurately assign the samples into two classes?

A: The idea behind the perceptron model is to use a reductionist approach to mimic how a single neuron in the brain works: it either fires or it does not.

Artificial Neurons and Perceptron Learning Rule

- Thus, *Rosenblatt's* **perceptron algorithm** can be summarized by the following steps:
 1. Initialize the weights to 0 or small random numbers
 2. For each training sample $\mathbf{x}^{(i)}$, compute the output value \hat{y} , then update the weights:

$$w_j := w_j + \Delta w_j$$

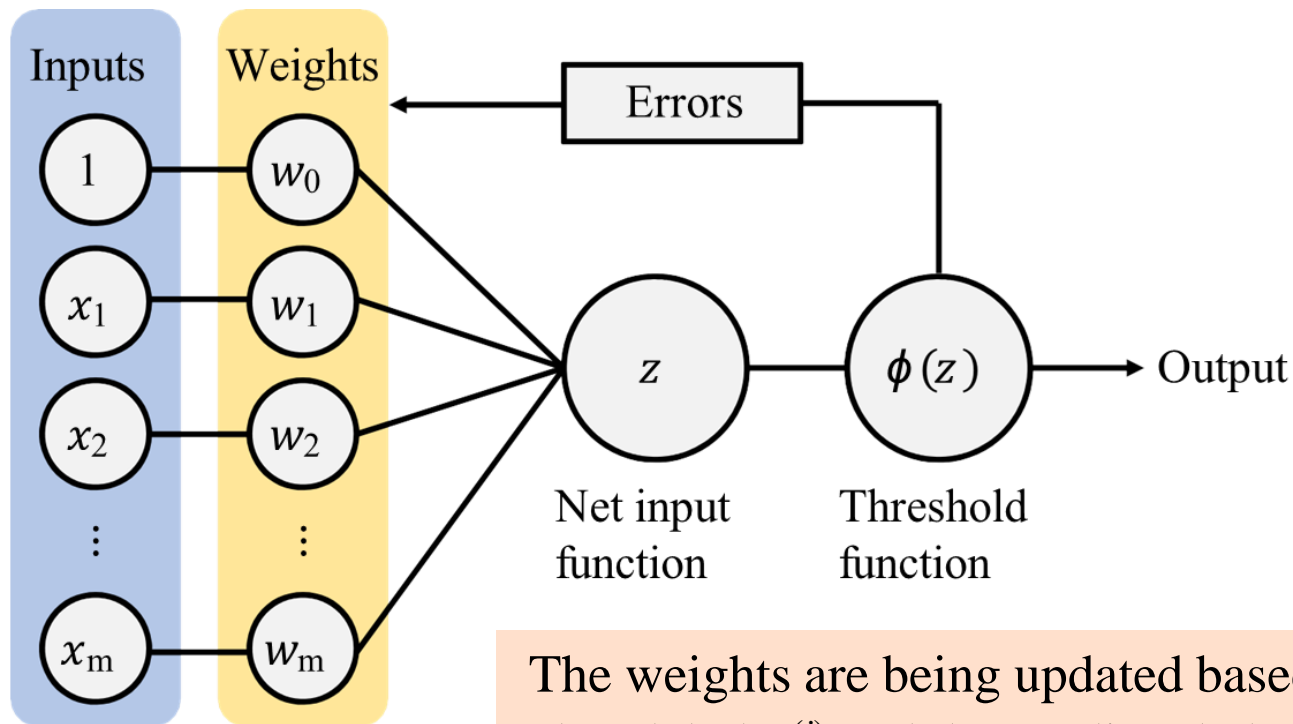
where the increment Δw_j for updating w_j is calculated by **perceptron learning rule**:

$$\Delta w_j = \eta (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

where **η is the learning rate (between 0 to 1)**, $y^{(i)}$ is the true class label and $\hat{y}^{(i)}$ is the predicted class label $\phi(z)$.

Artificial Neurons and Perceptron Learning Rule

- This algorithm is illustrated by the diagram below:



The weights are being updated based on the true class label $y^{(i)}$ and the predicted class label $\hat{y}^{(i)}$.

Artificial Neurons and Perceptron Learning Rule

- There are **three possible cases**:

1. Case I – correct prediction

- If $y^{(i)} = \hat{y}^{(i)}$, then no matter the class label is 1 or -1 , we have $\Delta w_j = 0$, no need to update the weights.

$$\Delta w_j = \eta(1 - 1)x_j^{(i)} = 0 \qquad \Delta w_j = \eta(-1 - (-1))x_j^{(i)} = 0$$

2. Case II – wrong prediction

- If $y^{(i)} = 1$ and $\hat{y}^{(i)} = -1$, then:

$$\Delta w_j = \eta(1 - (-1))x_j^{(i)} = 2\eta x_j^{(i)} > 0$$

which will push the weight towards the direction of the positive target class.

Artificial Neurons and Perceptron Learning Rule

3. Case III – wrong prediction

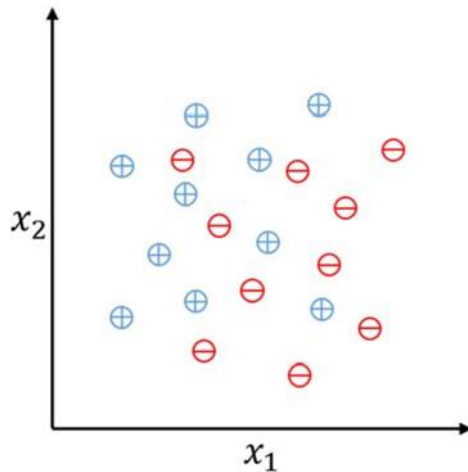
- If $y^{(i)} = -1$ and $\hat{y}^{(i)} = 1$, then:

$$\Delta w_j = \eta(-1 - 1)x_j^{(i)} = -2\eta x_j^{(i)} < 0$$

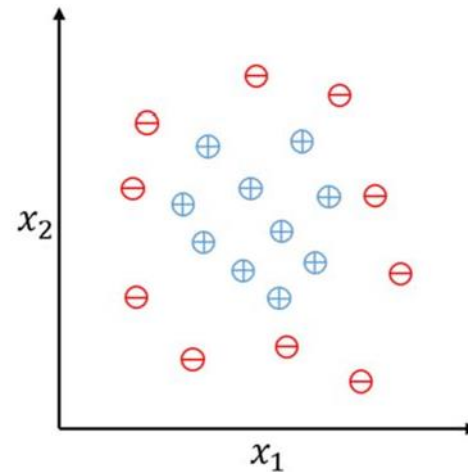
which will push the weight towards the direction of the negative target class.

Artificial Neurons and Perceptron Learning Rule

- Notice that the **convergence of the perceptron model** is only guaranteed if the two classes are linearly separable.
- The below figures show two examples of non-linearly separable points.




Two classes being highly overlapped and hard to separate.



Two classes being separable by a circle but not by a straight line.

Python Data Analytics

- We have introduced descriptive statistics for describing the characteristics of a set of data.
 - However, the computation can be tedious when the data size is large.
 - **Computer programming** can be used as a tool for making such computations.
 - In this chapter, we will introduce useful techniques for data analytics **using Python with Pandas (Python Data Analysis Library)**.
- 



Python Data Analytics

- **Pandas stand for Python Data Analysis.**
- It is a library that **contains data structures and data manipulation tools** designed to make data cleaning and analysis fast and easy in Python.
- Pandas adopts **array-based computing**, which allows data processing without for loops.
- It is designed for working with tabular or heterogeneous data.
- Before using the functions in Pandas library, we have to import it first.
- When we **call functions** in Pandas, we can start with "pd.".

```
import pandas as pd
```



Python Data Analytics



- In Pandas, data can be stored into two forms:
 - **Series**
 - **DataFrame**
 - Notice that the letters 'S', 'D' and 'F' are capital.
- A **Series** is a one-dimensional array-like object containing a sequence of values and an associated array of data labels, called its **index**.
- We can simply convert a list of numerical values into a Series by applying `pd.Series()`.

Python Data Analytics

- For example, we can convert the list of first 5 prime numbers into a Series.
- The **index by default** is 0 to 4 (left figure).

```
x = pd.Series([2,3,5,7,11])
```

```
x
0    2
1    3
2    5
3    7
4   11
dtype: int64
```

```
y = pd.Series([2,3,5,7,11],index=['a','b','c','d','e'])
```

```
y
a    2
b    3
c    5
d    7
e   11
dtype: int64
```

- However, we may also **rename the index** as another list of string or numbers (right figure).

Python Data Analytics

- One useful feature of Series is **vectorized computation**.
- If we apply an **arithmetic operation** between two Series, the entries of the corresponding indexes will be calculated, resulting a new Series.

x	2	3	5	7	11
index	0	1	2	3	4

y	10	20	30	40	50
index	0	1	2	3	4

x+y	12	23	35	47	61
index	0	1	2	3	4

Python Data Analytics

- If we apply an **arithmetic operation** between a Series and a number, the operation will be applied to each entry of the Series.

x	2	3	5	7	11
index	0	1	2	3	4

x*2	4	6	10	14	22
index	0	1	2	3	4

Python Data Analytics

- We can verify this result in Python.

```
import pandas as pd
x = pd.Series([2,3,5,7,11])
y = pd.Series([10,20,30,40,50])
```

```
x = pd.Series([2,3,5,7,11])
y = pd.Series([10,20,30,40,50])
```

x+y

0	12
1	23
2	35
3	47
4	61

dtype: int64

x*2

0	4
1	6
2	10
3	14
4	22

dtype: int64

Python Data Analytics

- With vectorized computation, we can apply **standardization** to the whole Series easily.

```
import pandas as pd
A = pd.Series([28, 29, 30, 31, 32])
Z = (A - 30) / 2 ** 0.5
```

```
A = pd.Series([28, 29, 30, 31, 32])
Z = (A - 30) / 2 ** 0.5
```

Z	
0	-1.414214
1	-0.707107
2	0.000000
3	0.707107
4	1.414214

Descriptive Statistics and Arithmetic in Pandas

- **Pandas Series** are equipped with a set of common mathematical and statistical methods.
- The table shows some **common descriptive statistics methods** in Pandas.
- To apply these methods, put a dot "." after a Series and then call the **functions**.

Statistical meaning	Function
sum	<code>sum()</code>
arithmetic mean	<code>mean()</code>
median	<code>median()</code>
maximum	<code>max()</code>
minimum	<code>min()</code>
mode	<code>mode()</code>

Statistical meaning	Function
sample size	<code>count()</code>
mean absolute deviation	<code>mad()</code>
population variance	<code>var(ddof=0)</code>
sample variance	<code>var(ddof=1)</code>
population standard deviation	<code>std(ddof=0)</code>
sample standard deviation	<code>std(ddof=1)</code>

Descriptive Statistics and Arithmetic in Pandas

- Using these functions, we can verify the results in the previous chapter.

```
import pandas as pd
```

```
A = pd.Series([28,29,30,31,32])
```

```
B = pd.Series([10,10,30,50,50])
```

```
A = pd.Series([28,29,30,31,32])  
B = pd.Series([10,10,30,50,50])
```

```
A.mean()
```

```
30.0
```

```
B.mean()
```

```
30.0
```

```
A.median()
```

```
30.0
```

```
B.median()
```

```
30.0
```

```
A.mad()
```

```
1.2
```

```
B.mad()
```

```
16.0
```

```
A.var(ddof=0)
```

```
2.0
```

```
B.var(ddof=0)
```

```
320.0
```

```
A.std(ddof=0)
```

```
1.4142135623730951
```

```
B.std(ddof=0)
```

```
17.88854381999832
```

Descriptive Statistics and Arithmetic in Pandas

- Instead of showing each particular item, we can also generate a **summarized result of the descriptive statistics of a Series** using the `describe()` method.

```
A.describe()
```

```
count    5.000000
mean     30.000000
std       1.581139
min      28.000000
25%      29.000000
50%      30.000000
75%      31.000000
max      32.000000
dtype: float64
```

```
B.describe()
```

```
count    5.0
mean     30.0
std      20.0
min      10.0
25%      10.0
50%      30.0
75%      50.0
max      50.0
dtype: float64
```

- Notice that the `std` here refers to sample standard deviation.
- The numbers resulted from the `describe` method are calculated based on the data values.
- However, some methods return the indirect statistics.

```
A.idxmax()
```

```
4
```

```
A.idxmin()
```

```
0
```

Descriptive Statistics and Arithmetic in Pandas

- For example, we know the maximum value in Series A is 32, but we might not know where this value is. The functions `idxmax()` and `idxmin()` returns the **index of the maximum/minimum entry** in a Series.

```
A.describe()
```

```
count    5.000000
mean     30.000000
std       1.581139
min      28.000000
25%      29.000000
50%      30.000000
75%      31.000000
max      32.000000
dtype: float64
```

```
B.describe()
```

```
count    5.0
mean     30.0
std      20.0
min      10.0
25%      10.0
50%      30.0
75%      50.0
max      50.0
dtype: float64
```

```
A.idxmax()
```

```
4
```

```
A.idxmin()
```

```
0
```

Descriptive Statistics and Arithmetic in Pandas

- In Series B, we find that there are **multiple entries with the same value**.
- If we want to check the unique values and the frequency of each unique value in a Series, we can use the function `value_counts()` which returns a new Series with its index being the unique values and its entries being the frequency of each unique values.
- This is what we call a **frequency table**.

```
B.value_counts()
```

```
10    2  
50    2  
30    1  
dtype: int64
```

Classwork



- The quiz score (out of 10) of 12 students are given below:

9, 8, 7, 9, 6, 5, 8, 2, 10, 8, 7, 5

Use Python to:

- a. Evaluate the mean value.
- b. Evaluate the sample standard deviation.
- c. Find the quartiles.
- d. Find the mode.
- e. Evaluate the population standard deviation.
- f. Create a frequency table.
- g. Standardize the data (use population standard deviation).

Checklist



- Can you:
 1. Describe how artificial neurons work and what the perceptron learning rule is?
 2. Discriminate all data points into two linearly separable classes?
 3. Discuss the three possible cases of perceptron algorithm?
 4. Apply descriptive statistics and arithmetic in Pandas?

