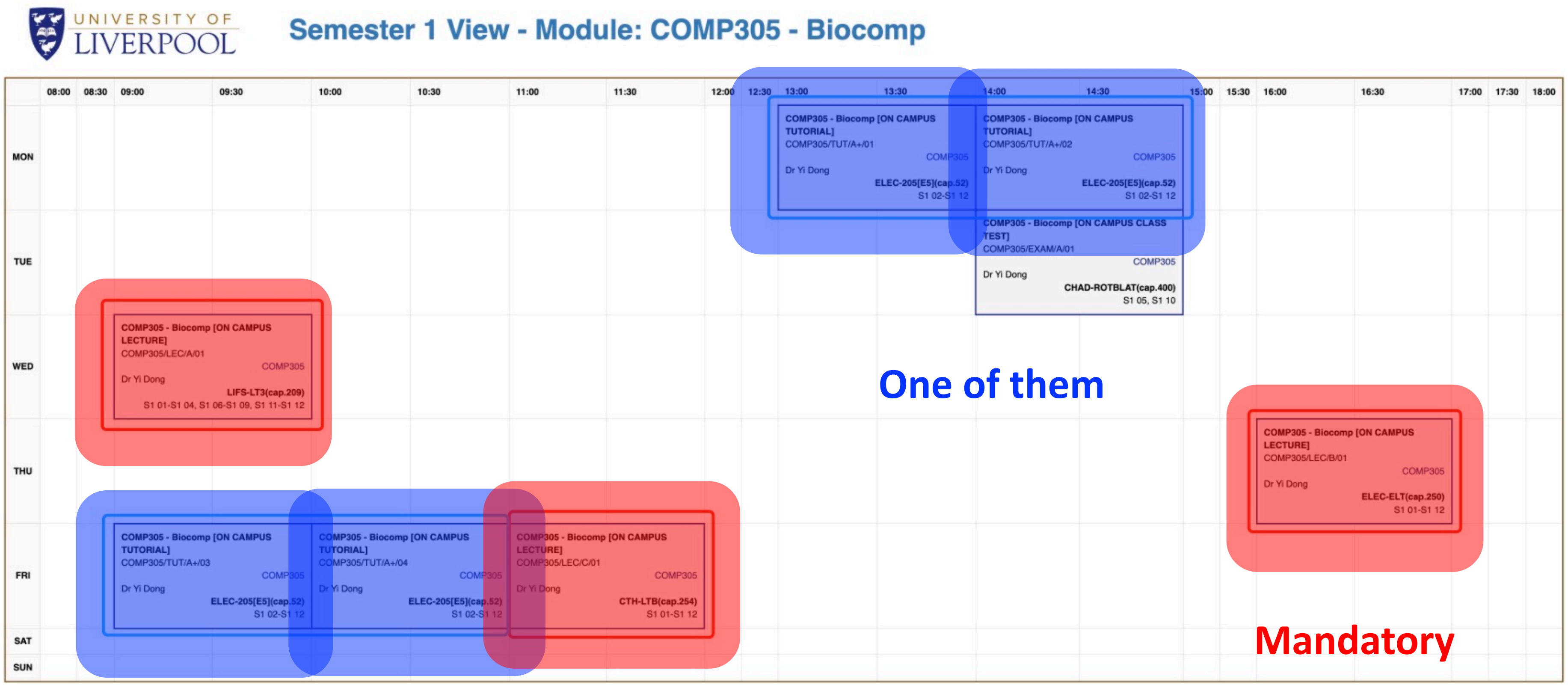
# Comp305

# Biocomputation

Lecturer: Yi Dong

# Comp305 Module Timetable




There will be **26-30** lectures, thee per week. The lecture slides will appear on Canvas. Please use Canvas to access the lecture information. There will be **9** tutorials, one per week.

# Lecture/Tutorial Rules

Questions are welcome as soon as they arise, because

1. Questions give feedback to the lecturer;

2. Questions help your understanding;

3. Your questions help your classmates, who might experience difficulties with formulating the same problems/doubts in the form of a question.

# Comp305 Part I.

# Artificial Neural Networks

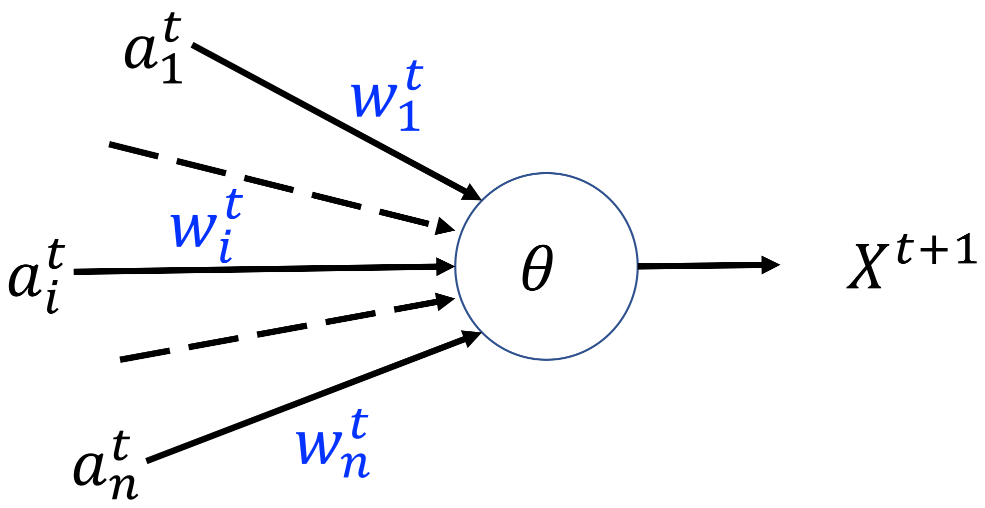
# ANN Learning Rules

- The ***McCullock-Pitts*** neuron made a base for a machine (network of units) capable of
  - ***storing information and***
  - ***producing logical and arithmetical operations on it***

- The next step
  - must be to realise another important function of the brain, which is

***to acquire new knowledge through experience*, i.e. *learning*.**

# ANN Learning Rules

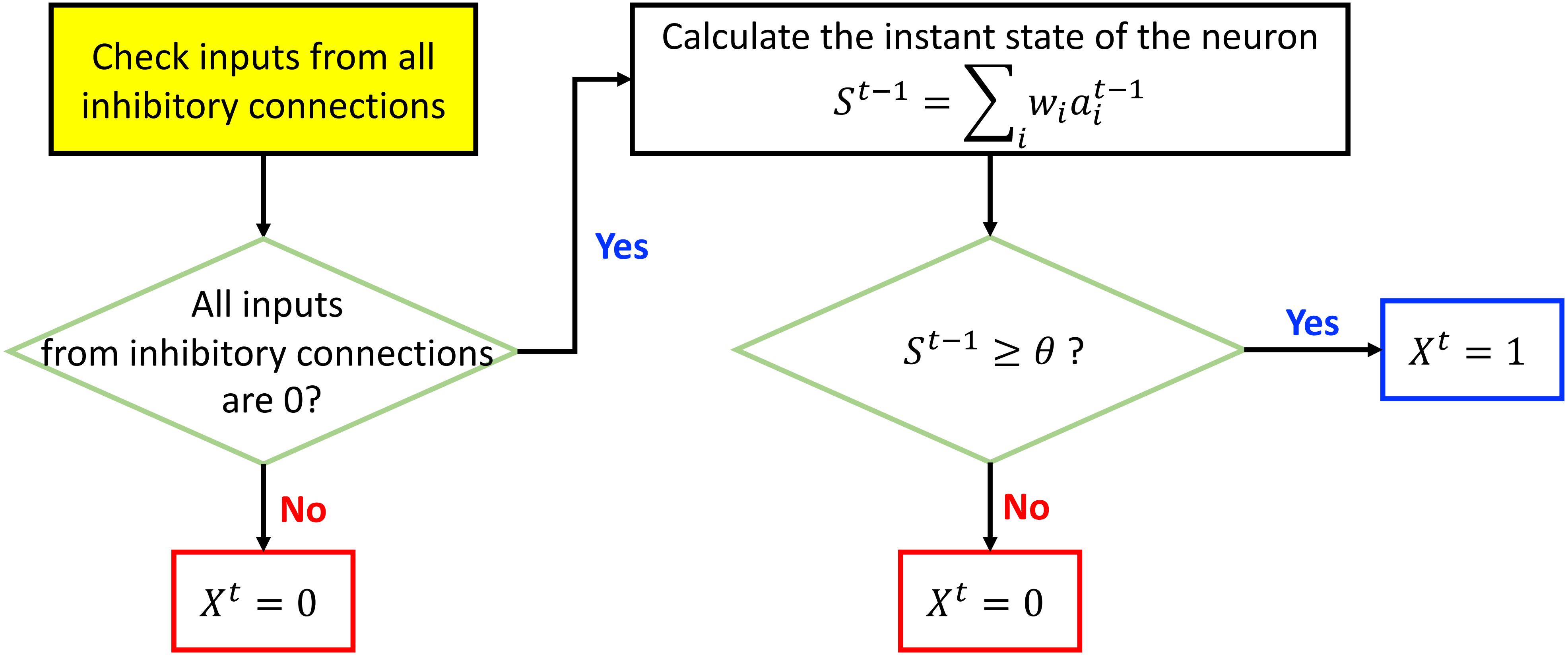



**Definition**:

**ANN learning rule is**
***the rule how to adjust the weights of connections to get desirable output.***

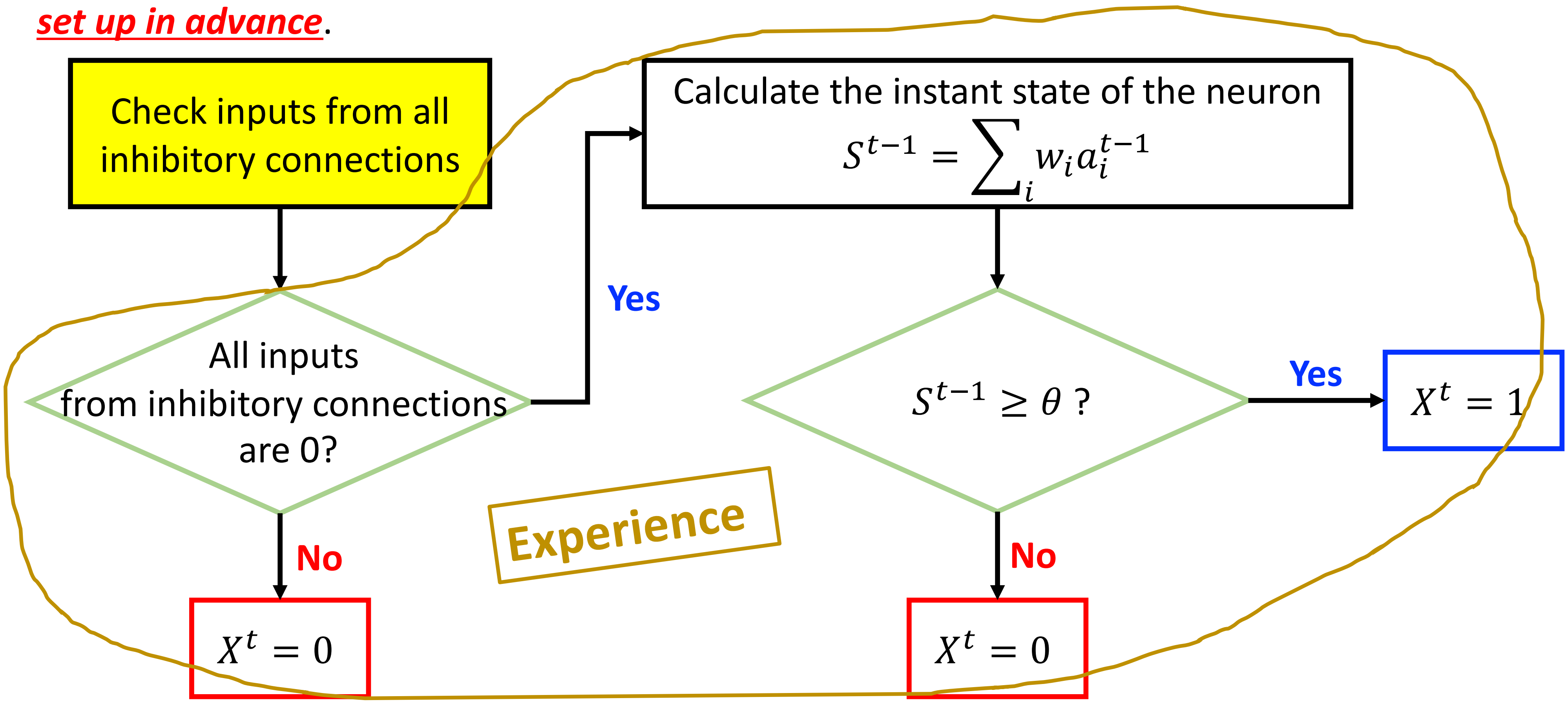
# MP Neuron vs. Hebb's Rule

Given a fixed MP neuron, that is, **all weights of connections** and neuron threshold are ***set up in advance***.



# MP Neuron vs. Hebb's Rule

Given a fixed MP neuron, that is, **all weights of connections** and neuron threshold are ***set up in advance***.



# MP Neuron vs. Hebb's Rule

1. Set the neuron threshold value $\theta$.
2. Set *random initial values* for the weights of connections $w_i^t$.
3. Give instant input values $a_i^t$ by the input units.
4. Compute the instant state of the neuron $S^t = \sum_i w_i^t a_i^t$
5. Compute the instant output of the neuron $X^{t+1}$
$$X^{t+1} = g(S^t) = H(S^t - \theta) = \begin{cases} 1, & S^t \geq \theta; \\ 0, & S^t < \theta. \end{cases}$$
6. Compute the instant corrections to the weights of connections $\Delta w_i^t = C a_i^t X^{t+1}$
7. Update the weights of connections $w_i^{t+1} = w_i^t + \Delta w_i^t$
8. **Go to the step 3**.

# MP Neuron vs. Hebb's Rule

1. Set the neuron threshold value $\theta$.
2. Set *random initial values* for the weights of connections $w_i^t$.
3. Give instant input values $a_i^t$ by the input units.
4. Compute the instant state of the neuron $S^t = \sum_i w_i^t a_i^t$
5. Compute the instant output of the neuron $X^{t+1}$

$$X^{t+1} = g(S^t) = H(S^t - \theta) = \begin{cases} 1, & S^t \geq \theta; \\ 0, & S^t < \theta. \end{cases}$$

**Experience**

6. Compute the instant corrections to the weights of connections $\Delta w_i^t = C a_i^t X^{t+1}$
7. Update the weights of connections $w_i^{t+1} = w_i^t + \Delta w_i^t$
8. **Go to the step 3**.

**Learning**

# Unsupervised Learning

- Unsupervised learning is a type of machine learning in which the algorithm **is not provided with any pre-assigned labels or scores for the training data**. As a result, unsupervised learning algorithms must first **self-discover any naturally occurring patterns** in that training data set.

- Hebb’s rule, Oja’s rule, Kohonen rule are all unsupervised learning rules.

# ANN Learning Rules

- As for the first time the problem was formulated in **1940s**, when experimental neuroscience was limited, the classic definitions of these learning rules came not from biology, but

from psychological studies of
**Donald Hebb** and **Frank Rosenblatt**.

**Unsupervised learning**

# ANN Learning Rules

- As for the first time the problem was formulated in **1940s**, when experimental neuroscience was limited, the classic definitions of these learning rules came not from biology, but

from psychological studies of

**Donald Hebb** and **Frank Rosenblatt**.

**Unsupervised learning**

**Supervised learning**

# Topic 6.

# Perceptron

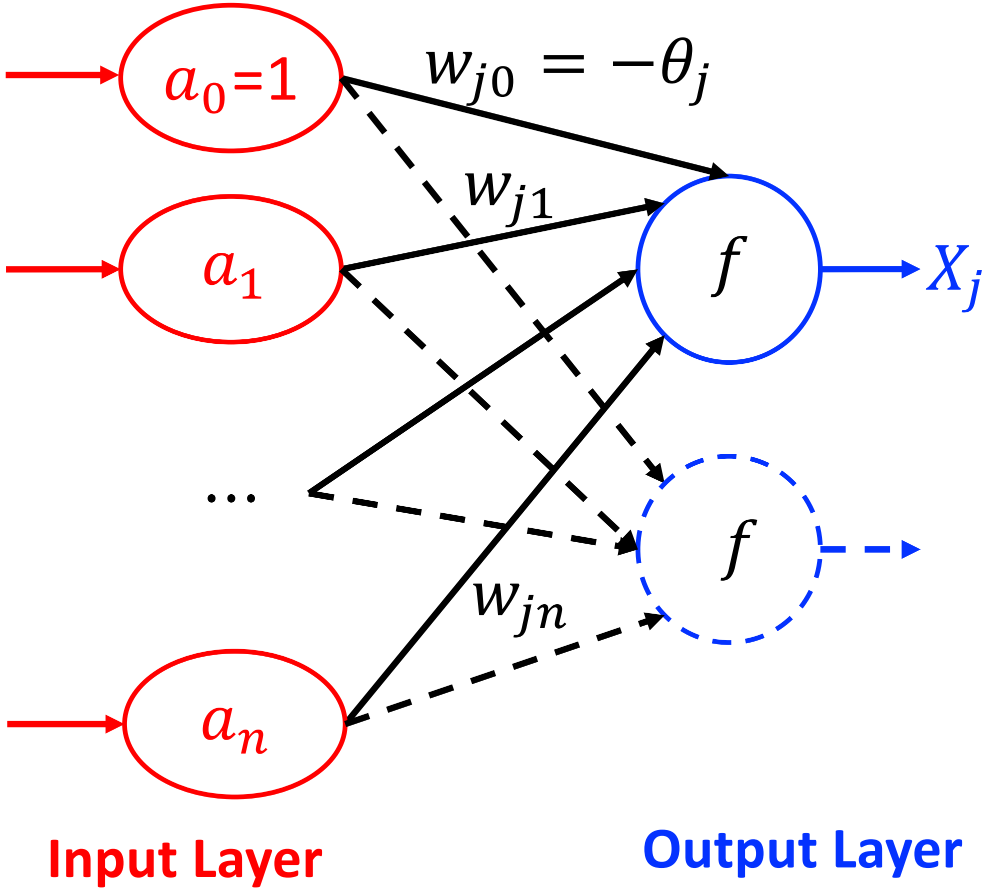
# Perceptron (1958)

- **Rosenblatt** (1958) explicitly considered the ***problem of pattern recognition***, where a *"teacher"* is essential. (Consider the difference with unsupervised learning)

- *A Perceptron* is a neural network that changes with "experience" using an ***error-correction rule*****.**

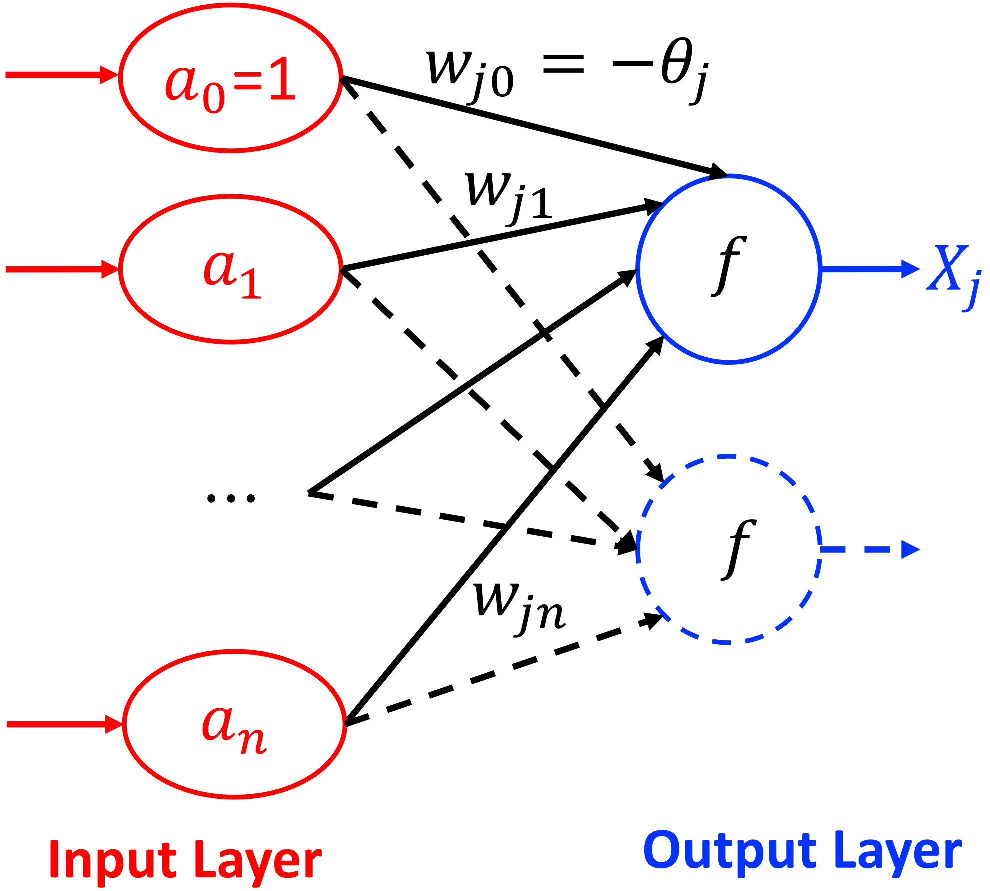
- According to the rule, **weight of a neuron changes when it makes error response to the input presented to the network.**

# Perceptron (1958)




The simplest architecture of perceptron comprises **one layer of idealised "neurons",** which we also sometimes call "units" of the network.

# Perceptron (1958): Syntax and Structure

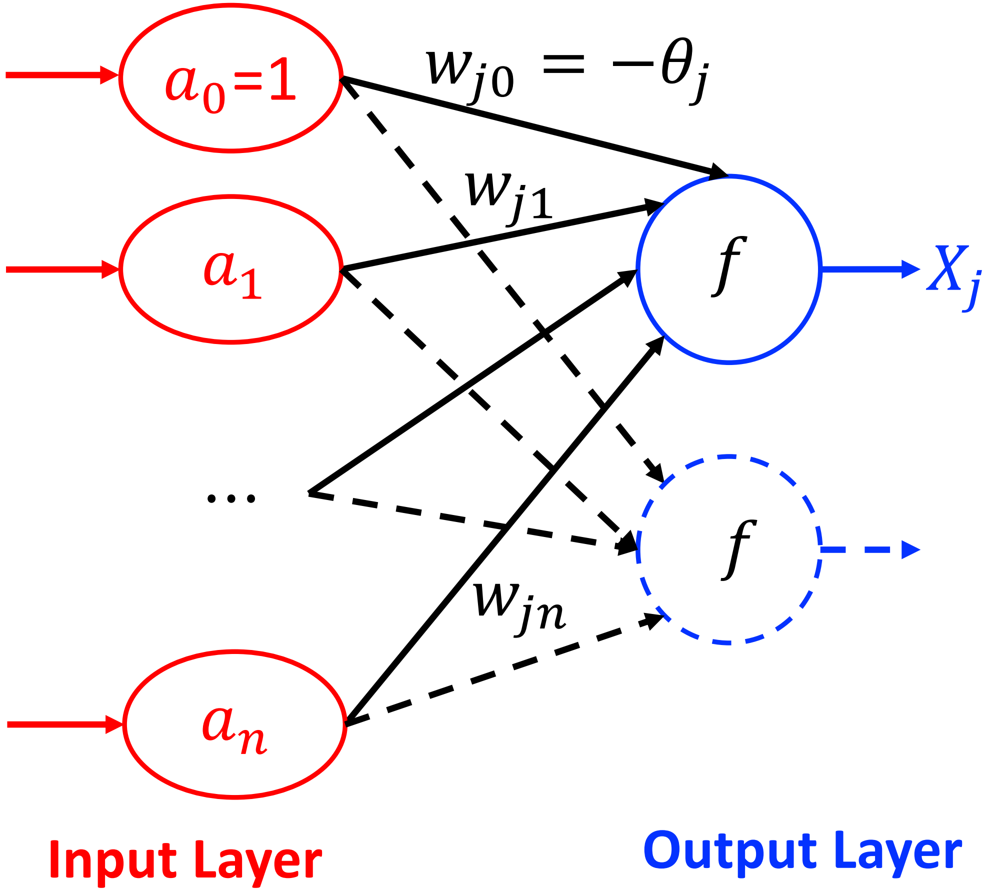



There are

- One layer of inputs
- One layer of output neurons

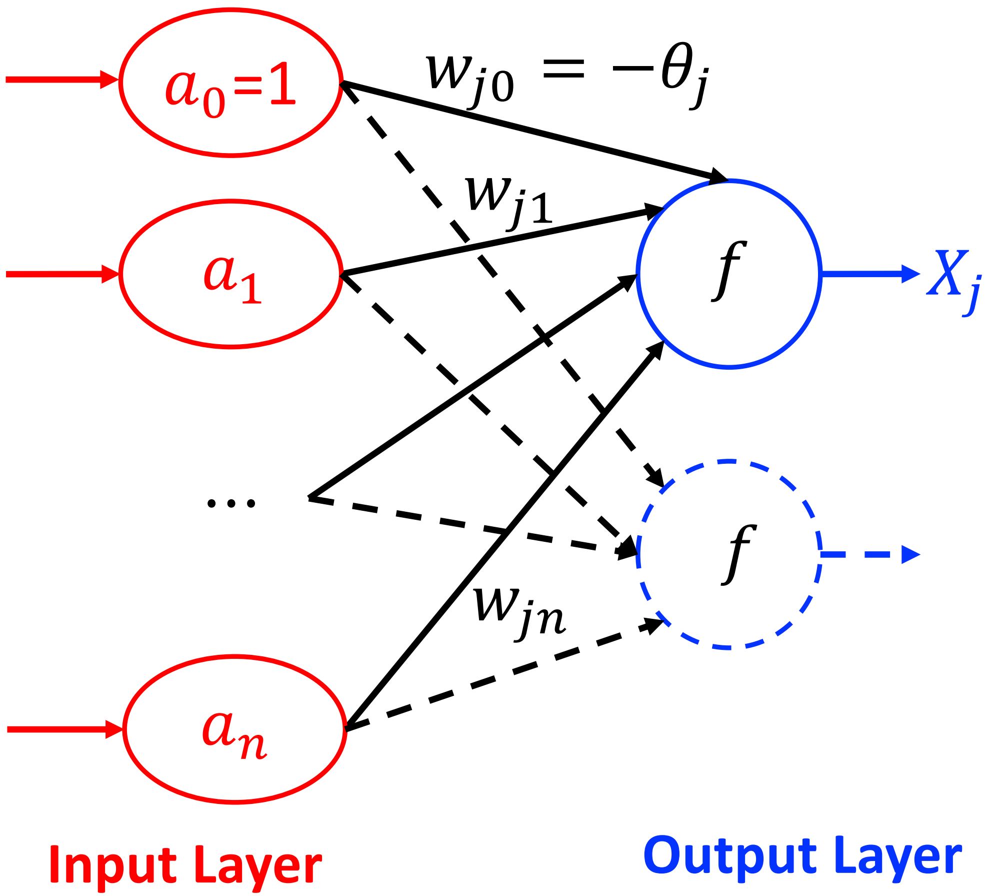
in the perceptron.

# Perceptron (1958): Syntax and Structure




The two layers are **fully interconnected**, *i.e.* ***every input neuron is connected to every output neuron***.

# Perceptron (1958): Syntax and Structure



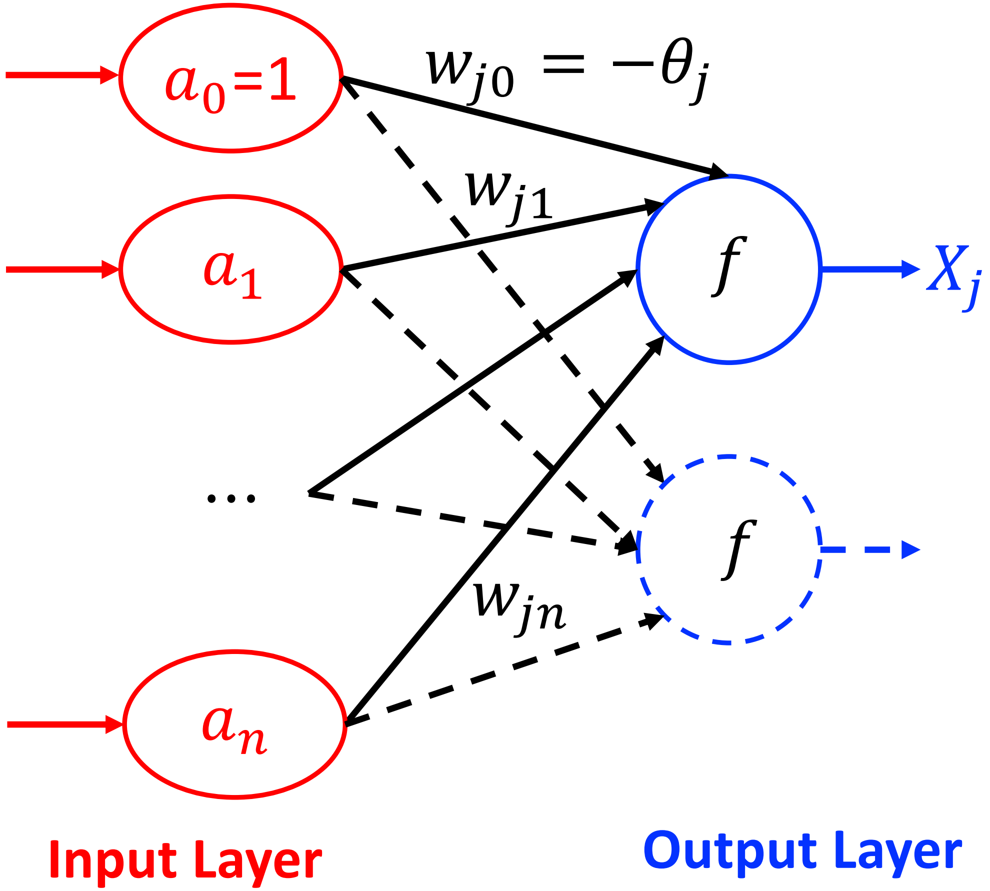

Semantically, a *perceptron* can be considered as a vector-valued function that maps the input

$$a = \{a_0, a_1, \cdots, a_n\}$$

to the output

$$X = \{X_1, X_2, \cdots, X_m\}$$

# Perceptron (1958): Syntax and Structure




Semantically, a *perceptron* can be considered as a vector-valued function that maps the input

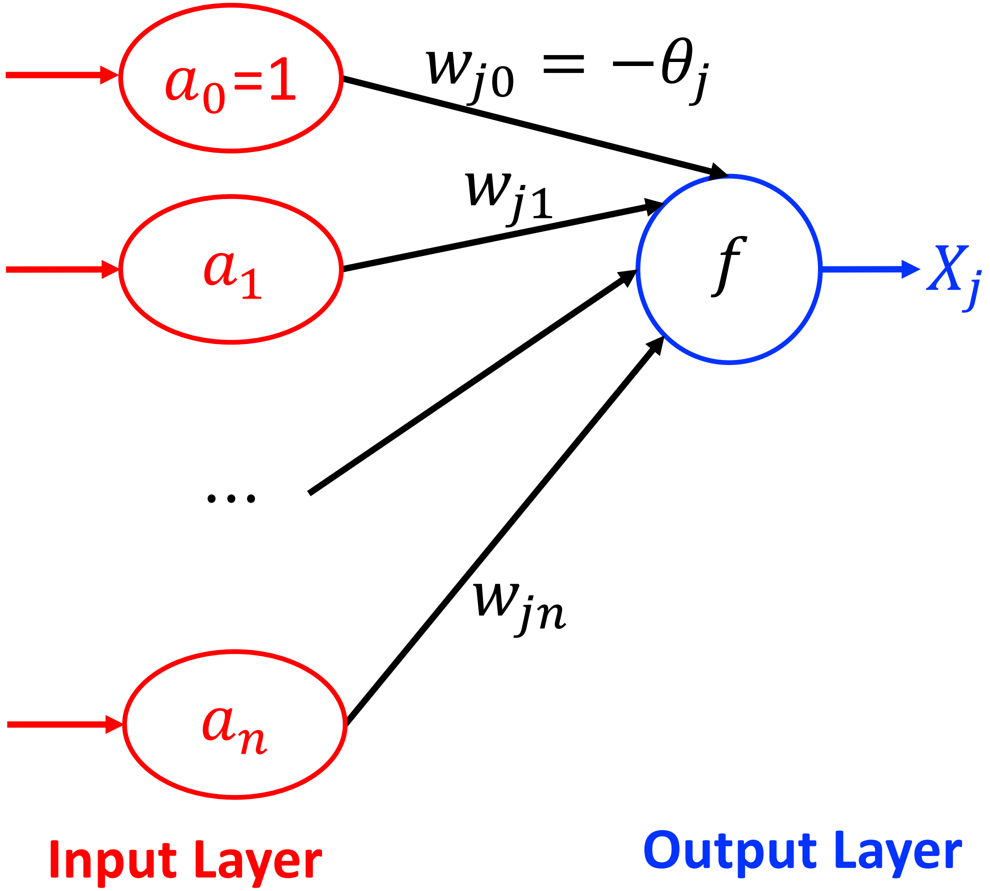
$$a = \{a_0, a_1, \cdots, a_n\}$$

to the output

$$X = \{X_1, X_2, \cdots, X_m\}$$

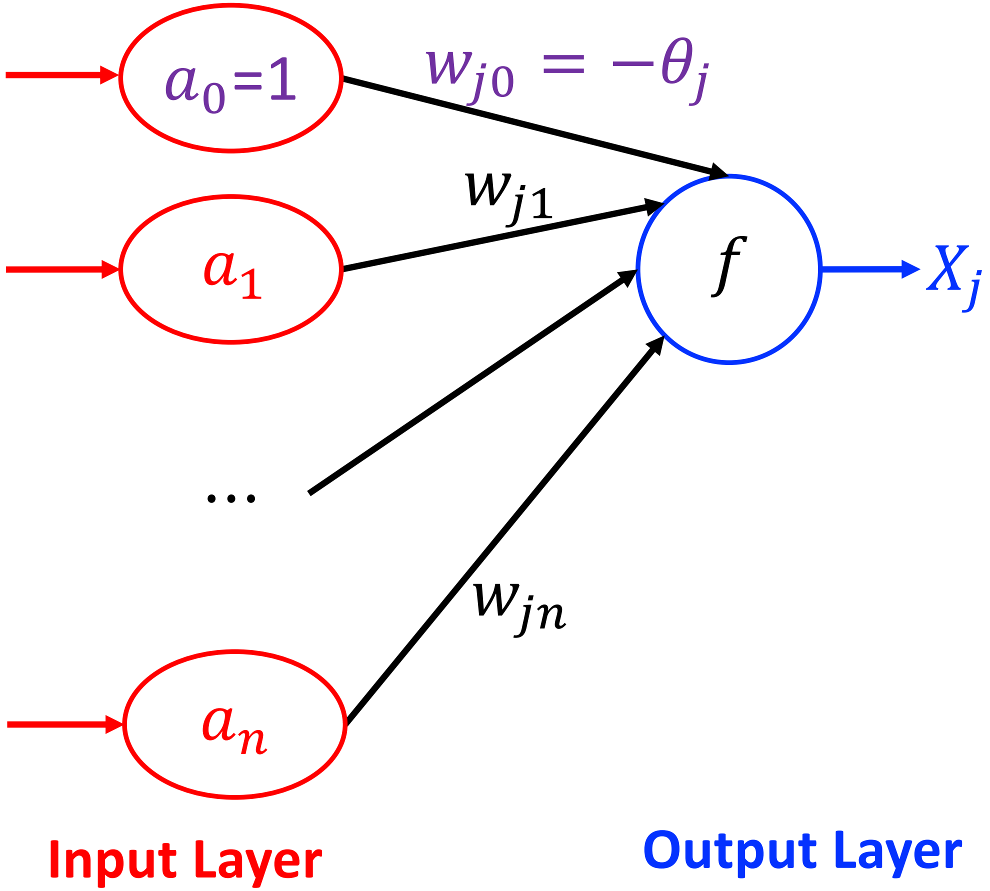
Similar to the model considered in Kohonen rule, a perceptron allows real inputs.

# Perceptron (1958): Syntax and Structure




Each output neuron has the **same inputs,** that is, the total input layer. But individual outputs are with **individual connections and therefore have different weights** of connections. Thus, we can consider each output independently. For instance, let us consider the $j$-th output neuron.

# Perceptron (1958): Semantics

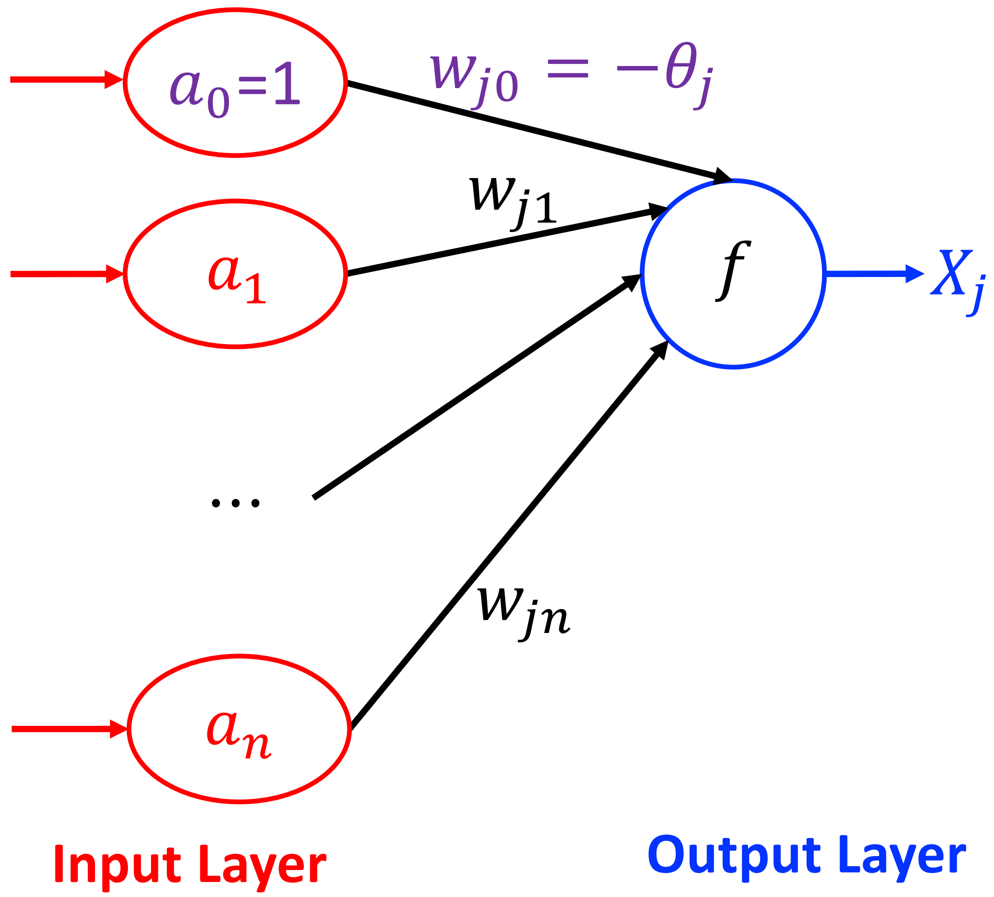



The weighted input to the $j$-th output neuron is

$$S_j = \sum_{i=0}^{n} w_{ji} a_i \,,$$

$a_0$ is a special input neuron with fixed input value of +1.
If we rename the weight of $w_{j0}$ for the $j$-th output as $-\theta_j$…

# Perceptron (1958): Semantics




The weighted input to the $j$-th output neuron is

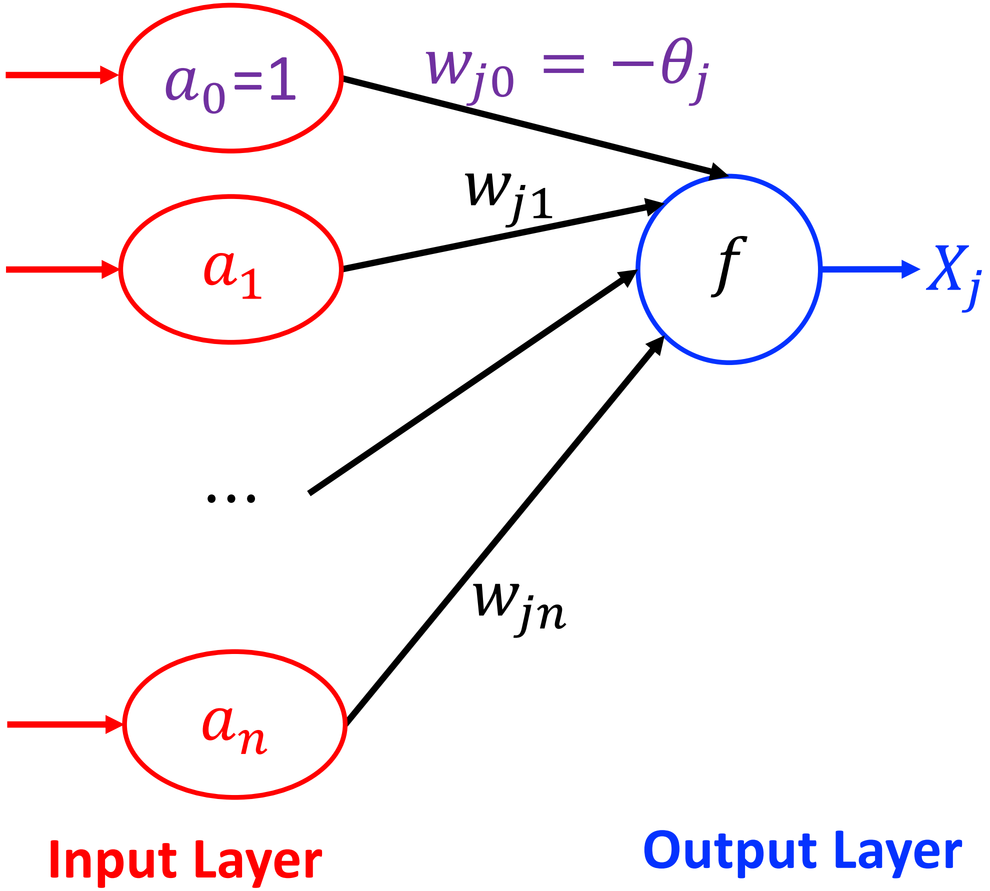
$$S_j = \sum_{i=0}^{n} w_{ji} a_i \,,$$

$a_0$ is a special input neuron with fixed input value of +1.

$$S_j = w_{j0} a_0 + \sum_{i=1}^{n} w_{ji} a_i$$

$$= -\theta_j + \sum_{i=1}^{n} w_{ji} a_i$$

# Perceptron (1958): Semantics



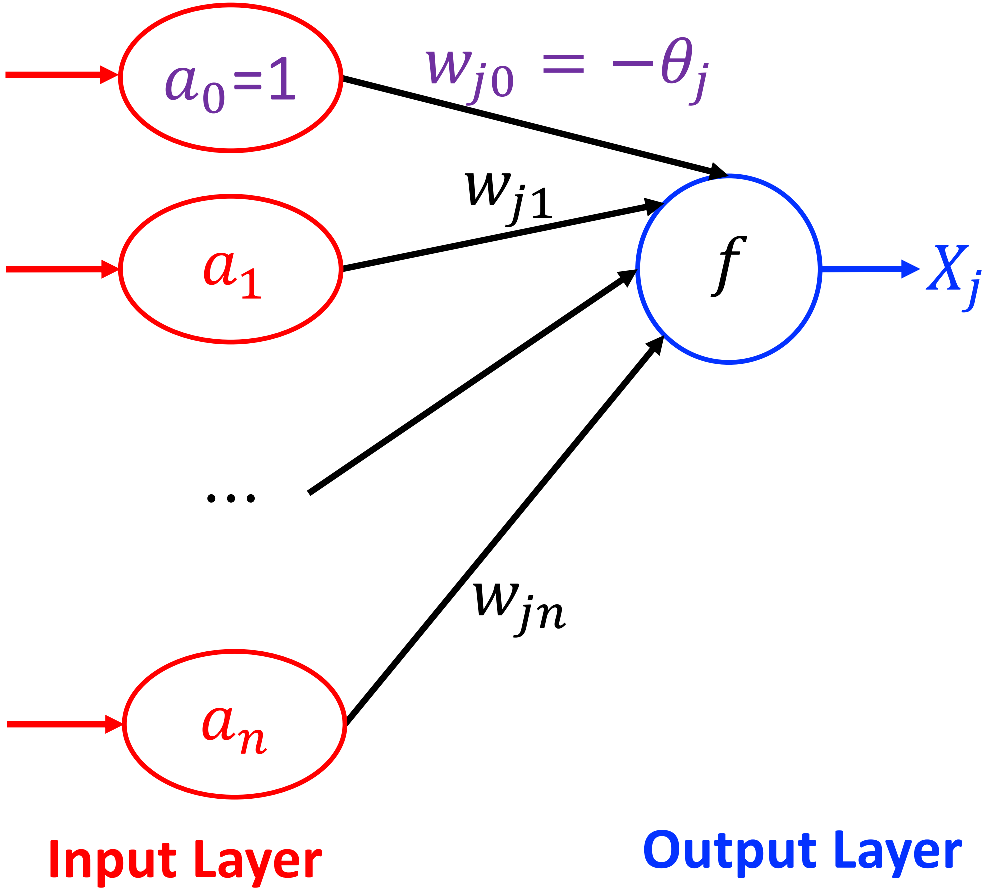

The weighted input to the $j$-th output neuron is

$$S_j = \sum_{i=0}^{n} w_{ji} a_i ,$$

$a_0$ is a special input neuron with fixed input value of +1.

With this trick, we have no need to set a threshold manually. Now we can **train the threshold like weights**.

# Perceptron (1958): Semantics

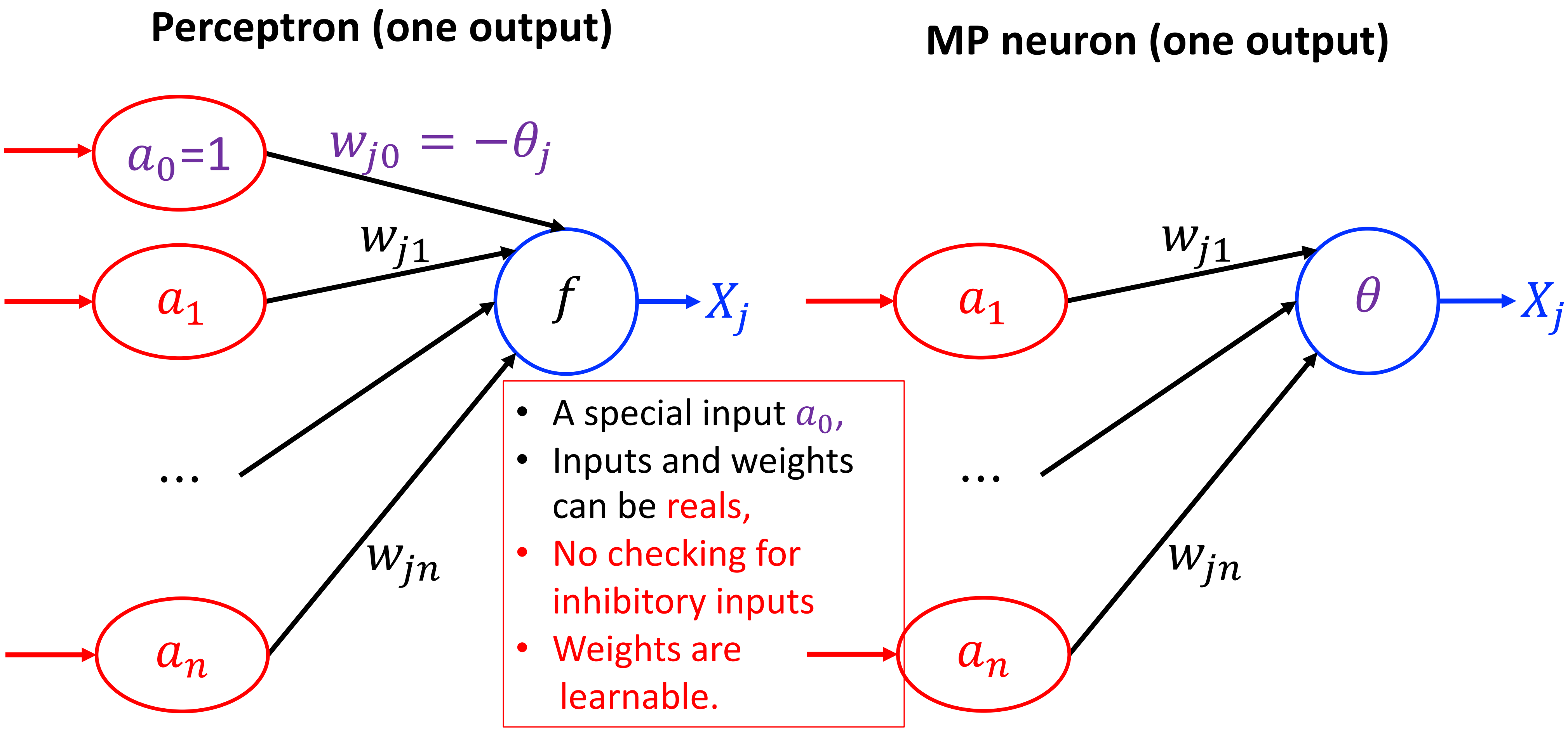



The value $X_j$ of $j$-th output neuron depends on whether the weighted input is greater than 0.

$$X_j = f(S_j) = \begin{cases} 1, & S_j \geq 0, \\ 0, & S_j < 0. \end{cases}$$

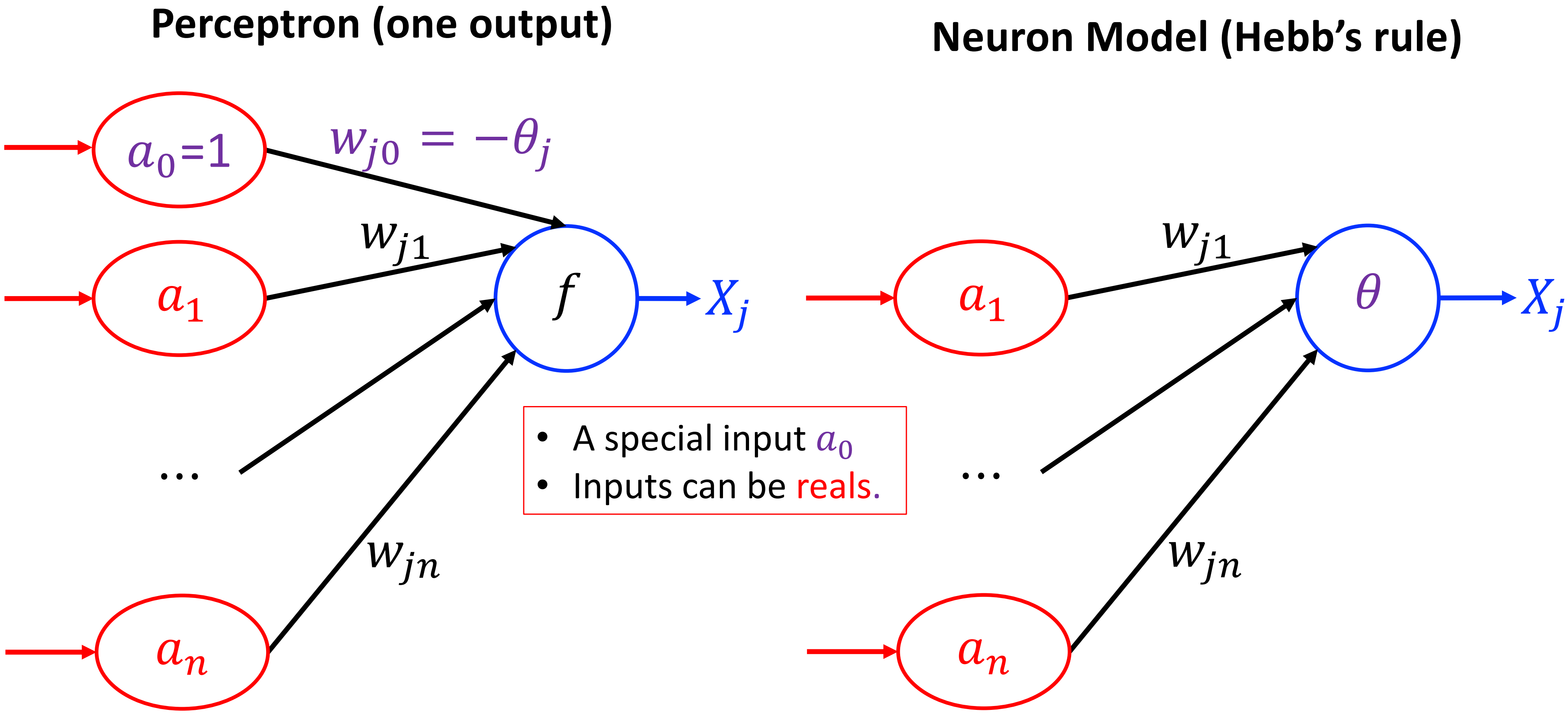
We call $f$ as ***activation function***.

# MP Neuron vs. Perceptron (Syntax and Semantics)




- A special input $a_0$,
- Inputs and weights can be reals,
- No checking for inhibitory inputs
- Weights are learnable.

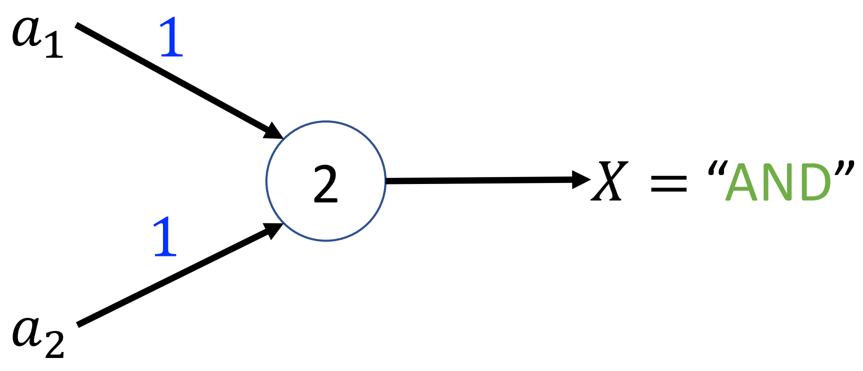
# Neuron Model in Hebb's Rule vs. Perceptron




- A special input $a_0$
- Inputs can be reals.

# Recall MP Neuron

| $\boldsymbol{a_1}$ | $\boldsymbol{a_2}$ | **"AND"** |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 0 |




**MP neuron can only "describe", but not "learn".**

# Recall Unsupervised Learning

$w_i^{t+1} = w_i^t + \Delta w_i^t,$
Where
$\Delta w_i^t = C a_i^t X^{t+1}$

**Hebb's Rule**

$w_i^{t+1} = w_i^t + \Delta w_i^t,$
Where
$\Delta w_i^t = C a_i^t X^{t+1}$
**Normalization:**
$\|\boldsymbol{w}^t\| = 1$

**Oja's Rule**

- The $j^*$-th output neuron has maximum weighted input at that instant $t$.

$$S_{j^*}^t = \max(S_1^t, \cdots, S_m^t)$$

- Then the weight of the $j$-th neuron is updated as

$$w_{ji}^{t+1} = w_{ji}^t + \Delta w_{ji}^t$$

where $i = 1, \cdots, n$, for $j = 1, \cdots, m.$

- The incremental term $\Delta w_{ji}^{t+1}$:

$$\Delta w_{ji}^{t+1} = C(t)\left(a_i^t - w_{ji}^t\right)\theta(j, j^*)$$

Where $\theta(j, j^*)$ is a restraint function due to the distance between neuron $j$ and $j^*$.

**Kohonen Rule**

**Unsupervised learning can only learn "similarity", but not predict "desired output".**

# What can a Perceptron be used for?

- Similar to Hebb's rule, we adjust **weights** (training) between two layers to learn knowledge from a given data set.

- If the data set is unlabeled, we can train the perceptron network to cluster the inputs to different groups (unsupervised learning).

# What can a Perceptron be used for?

- Similar to Hebb's rule, we adjust **weights** (training) between two layers to learn knowledge from a given data set.

- If the data set is unlabeled, we can train the perceptron network to cluster the inputs to different groups (unsupervised learning).

- If the data set is labeled, we can train the perceptron network to produce the desired output in response to certain inputs (supervised learning).