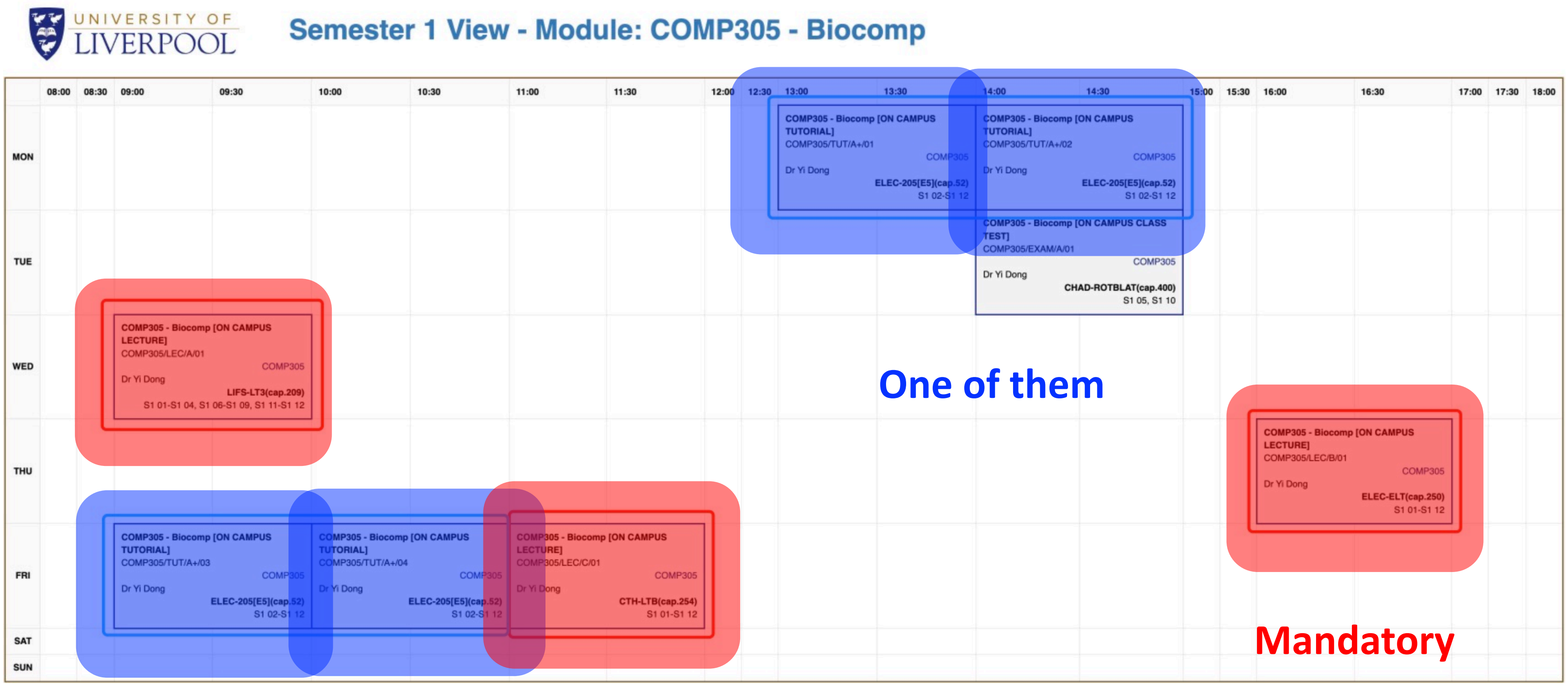
Comp305

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.

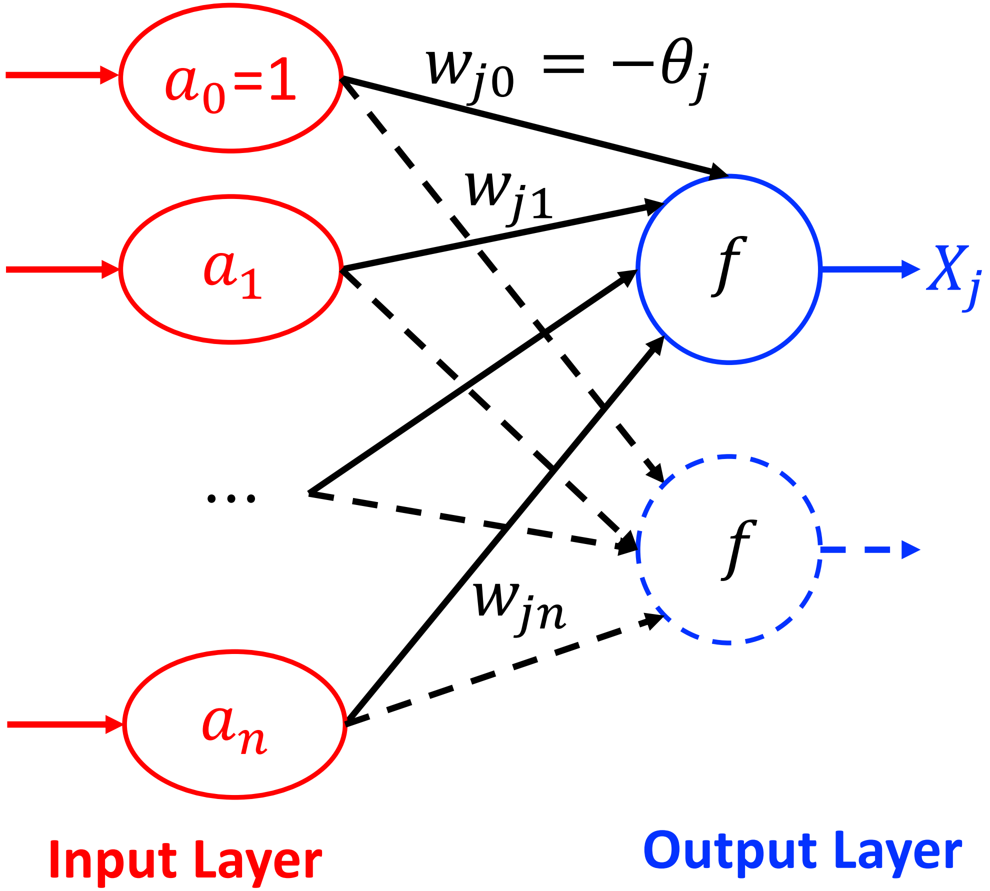
Comp305 Part I.

Artificial Neural Networks

Topic 6.

Perceptron

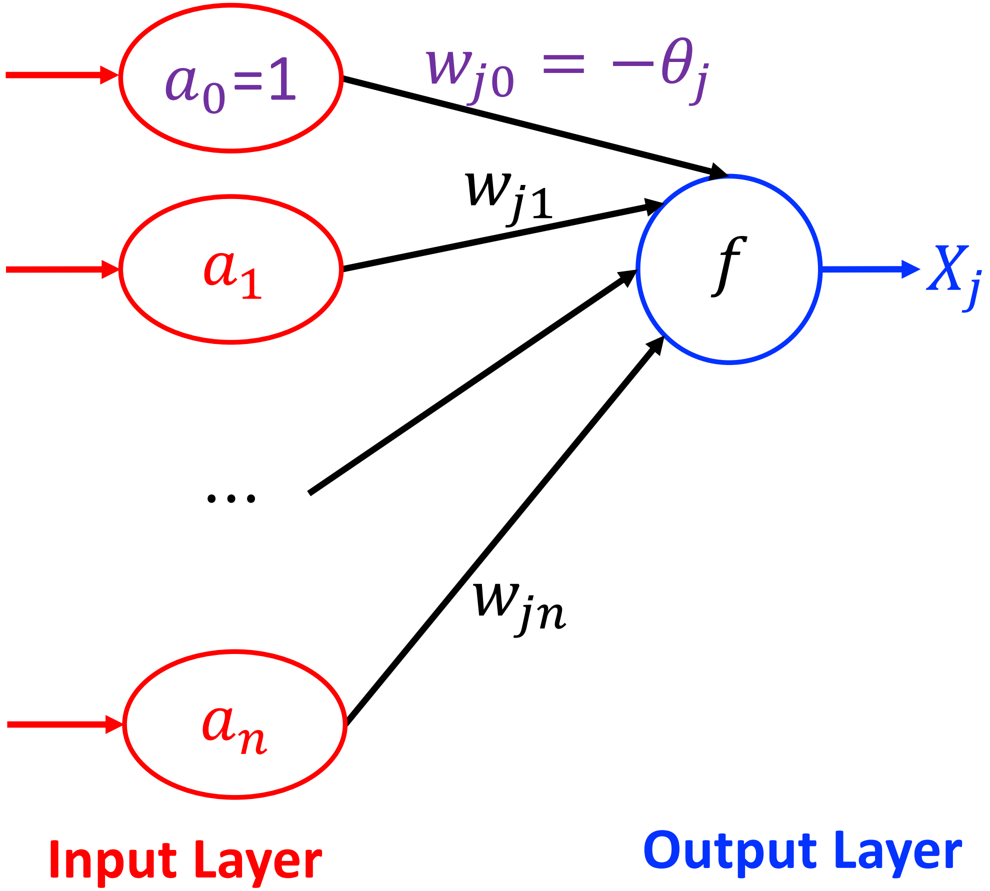
Perceptron (1958): Syntax and Structure



In a perceptron,

- One layer of input neurons
  - Real input value
  - a_0 is always 1
- One layer of output neurons
  - Binary output value
- fully interconnected architecture
- Each output neuron works independently

Perceptron (1958): Semantics



The weighted input to the j-th output neuron is

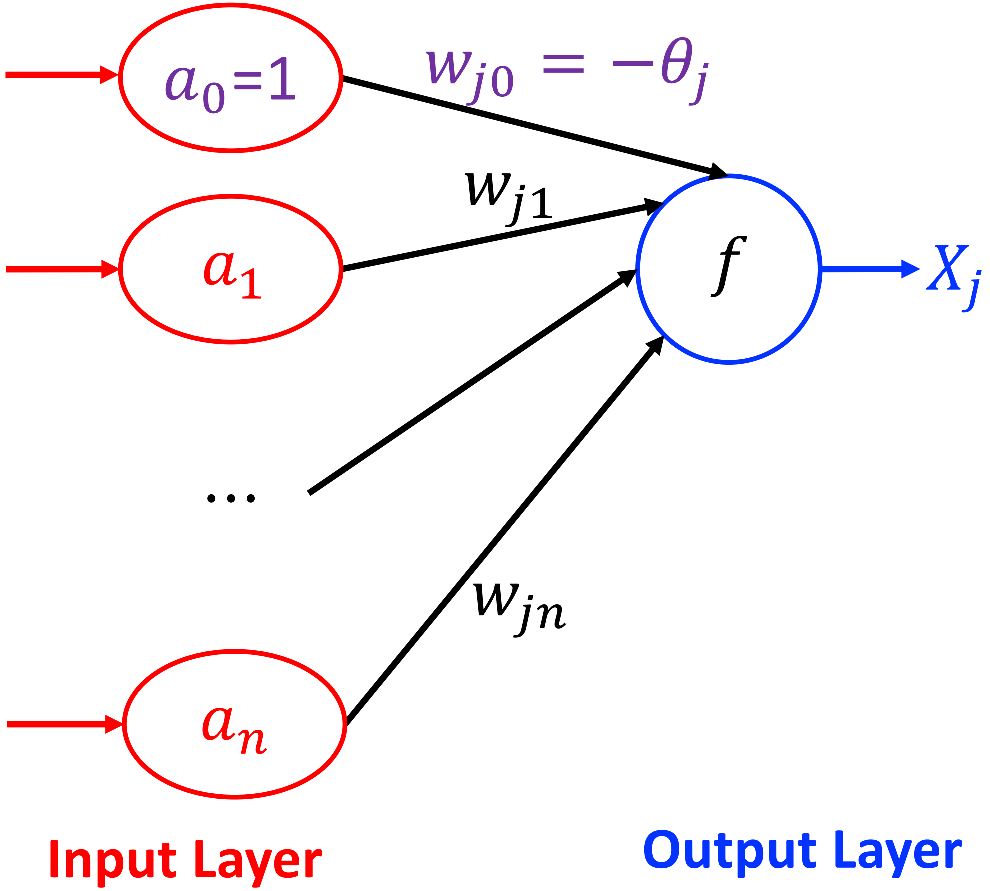
$$S_j = \sum_{i=0}^{n} w_{ji} a_i \,,$$

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 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***.

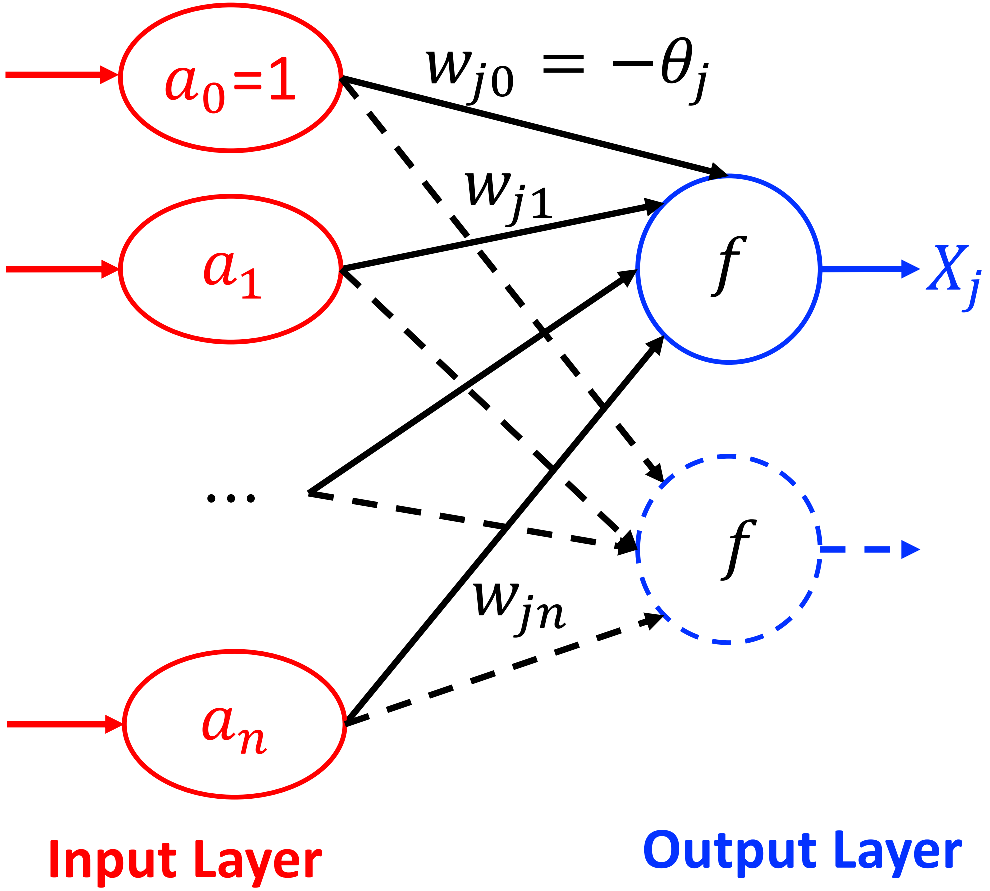
Perceptron (1958)

- **Rosenblatt** (1958) explicitly considered the ***problem of pattern recognition***, where a *“teacher”* is essential.

- *A Perceptron* is a neural network that changes with “experience” using an ***error-correction rule*. (Supervised learning!)**

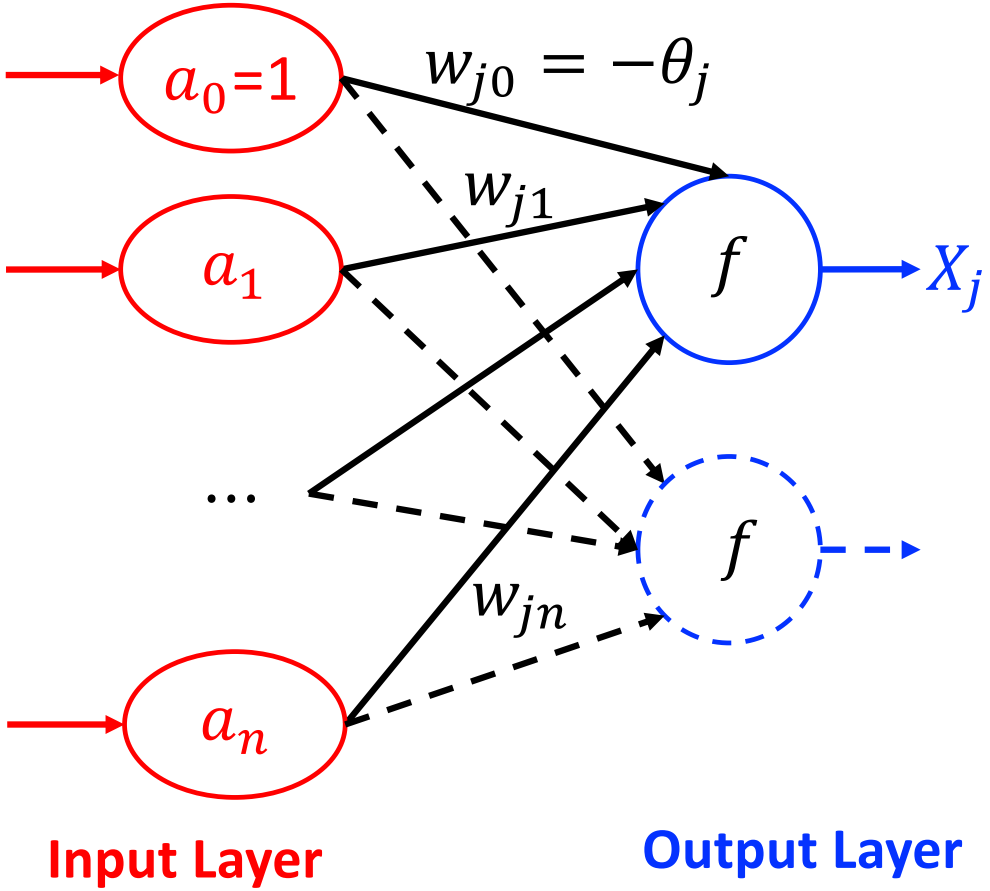
- According to the rule, **weight of a response unit changes when it makes error response to the input presented to the network.**

Perceptron (1958)



Weights w_{ji} of connections between two layers are changed according to perceptron learning rule, so the network is more likely to produce the desired output in response to certain inputs.

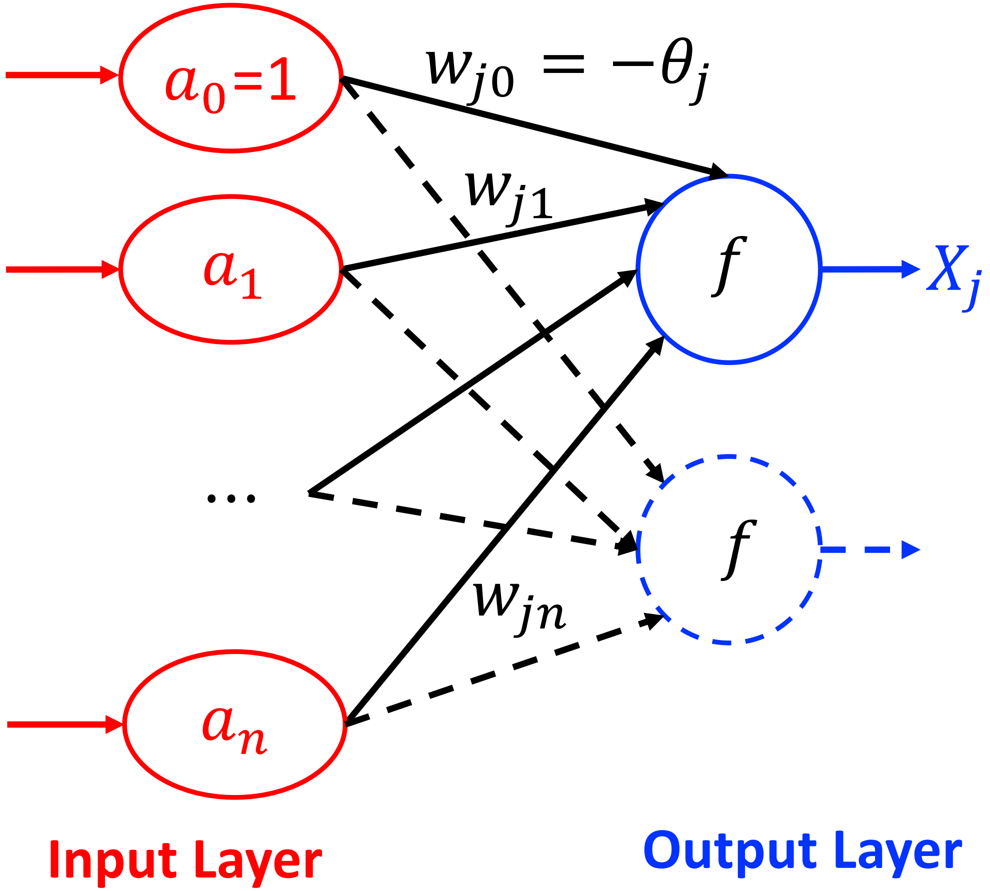
Perceptron (1958)



Weights w_{ji} of connections between two layers are changed according to perceptron learning rule, so the network is more likely to produce the desired output in response to certain inputs.

***The process of weights adjustment* is called <u>perceptron "learning" or "training"</u>.**

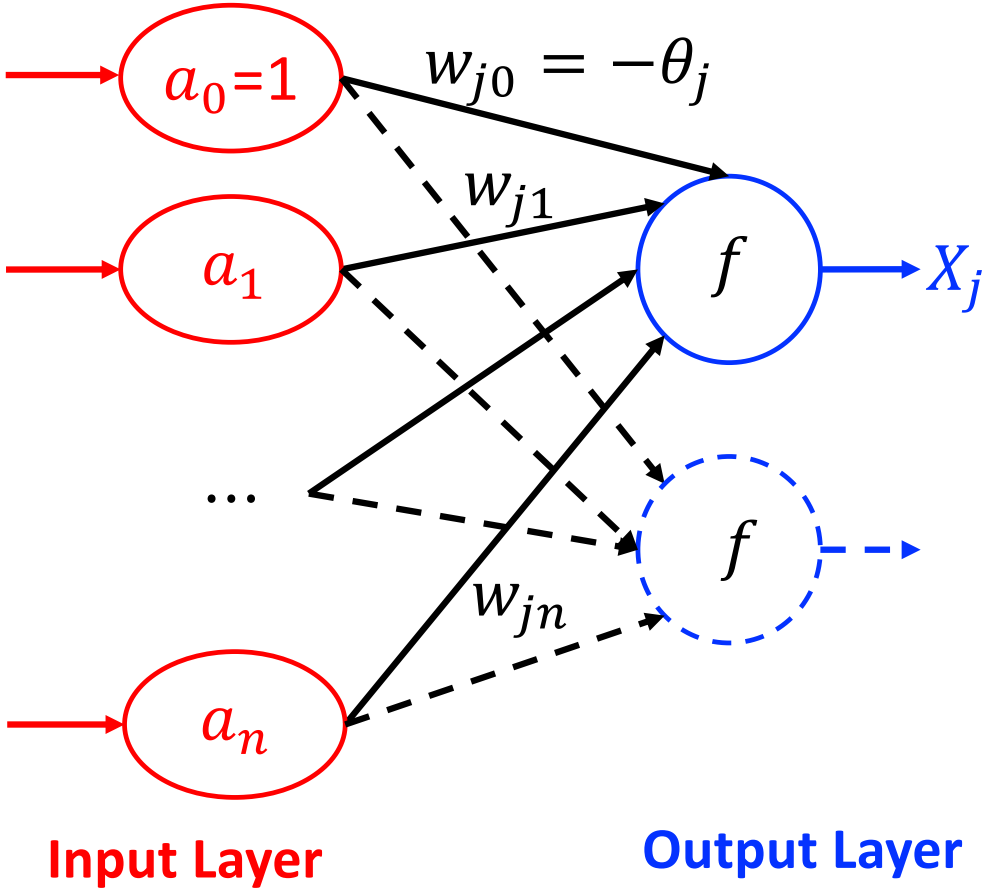
Perceptron Training



The perceptron is trained by using a ***training set*** with ***target outputs (labels)***.

- Training set: a **set of input patterns** repeatedly presented to the network during training;
- Target/desired output (label): the pre-defined correct output of an input pattern in the training set.

Perceptron Training

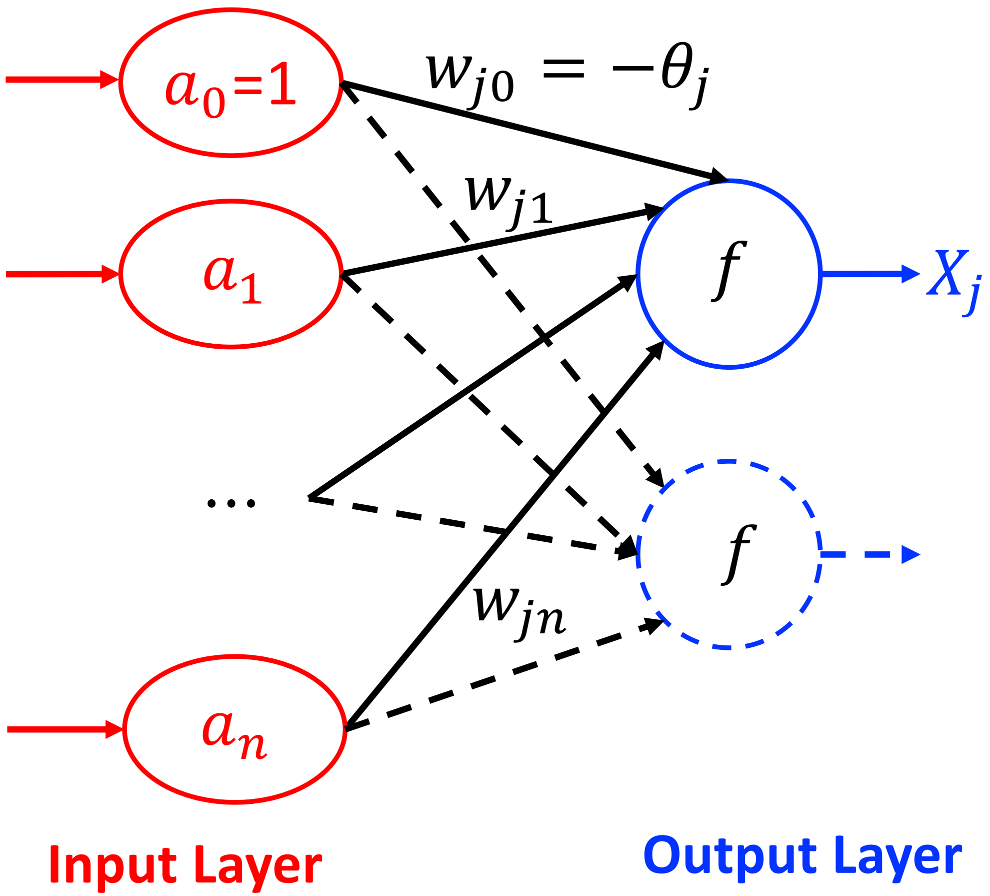


The perceptron is trained by using a *training set* with *target outputs (labels)*.

Every input pattern is used multiple times for training.

- Training set: a **set of input patterns** repeatedly presented to the network during training;
- Target output (label): the pre-defined correct output of an input pattern in the training set.

Perceptron Training

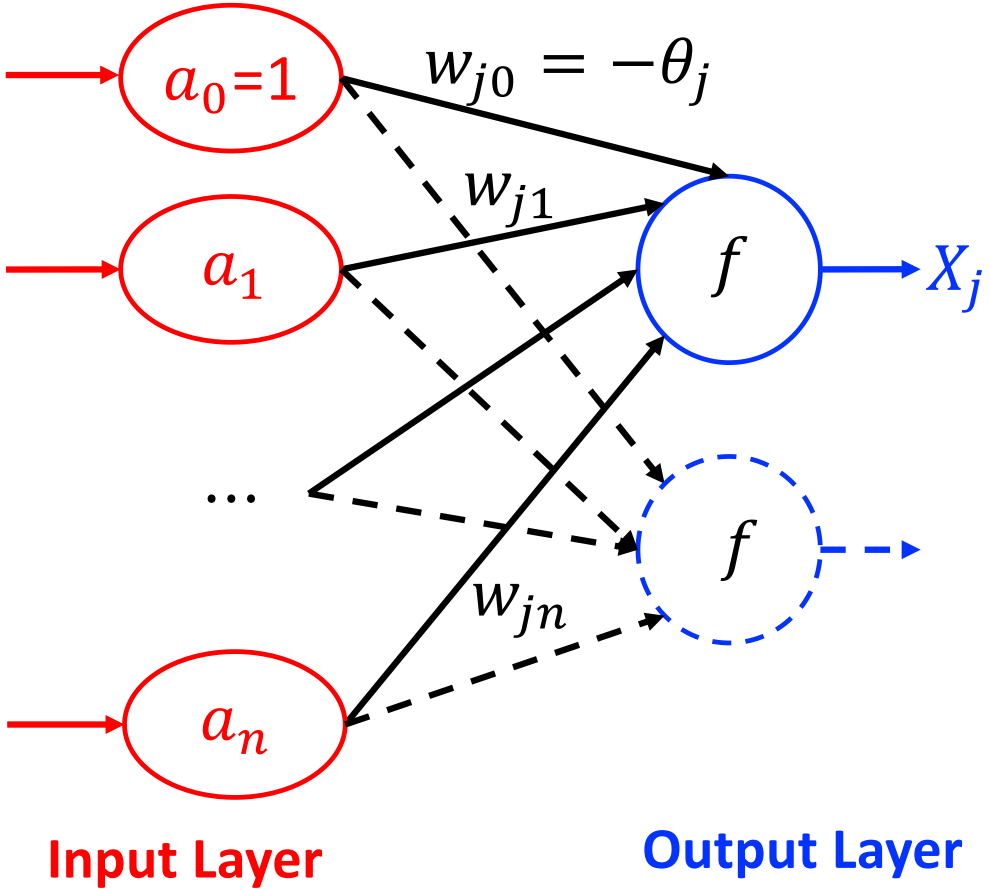


The perceptron is trained by using a ***training set*** with ***target outputs (labels)***.

Input pattern in the set is n-dimensional! Note a_0=1.

- Training set: a **set of input patterns** repeatedly presented to the network during training;
- Target output (label): the pre-defined correct output of an input pattern in the training set.

Perceptron Training

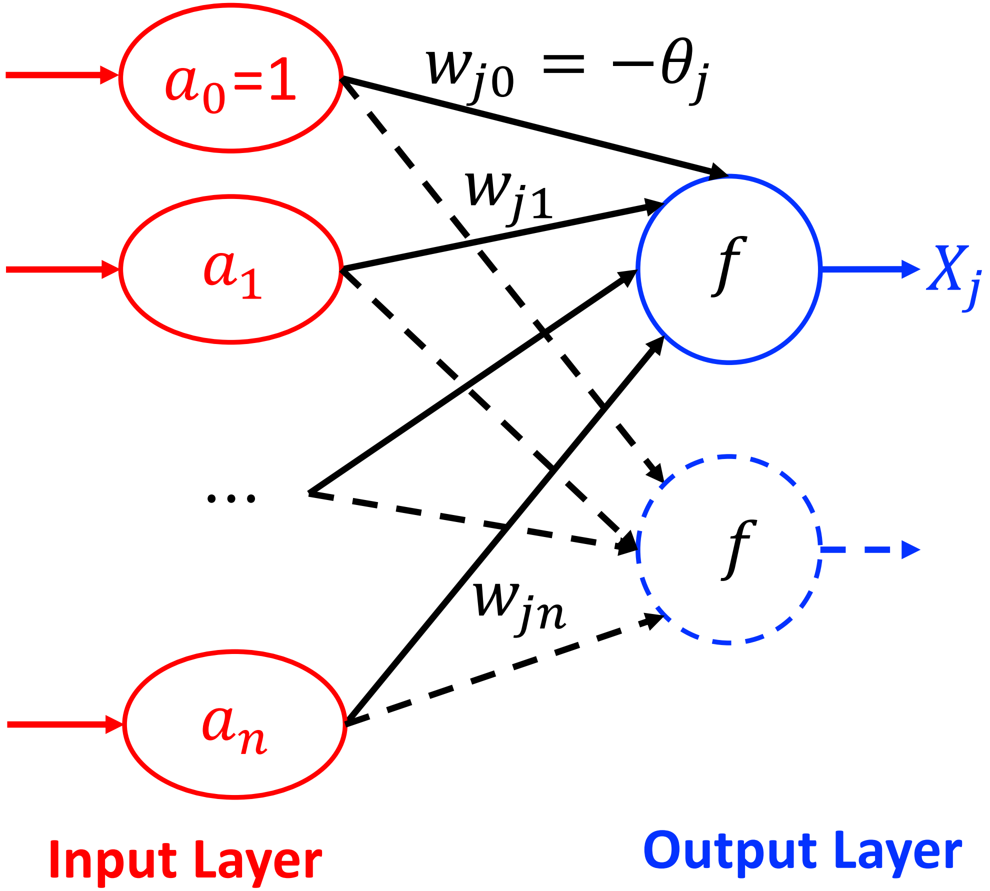


The perceptron is trained by using a ***training set*** with ***target outputs (labels)***.

- Training set: a **set of input patterns** repeatedly presented to the network during training;
- Target output (label): the pre-defined correct output of an input pattern in the training set.

Note that in unsupervised learning, no label is needed.

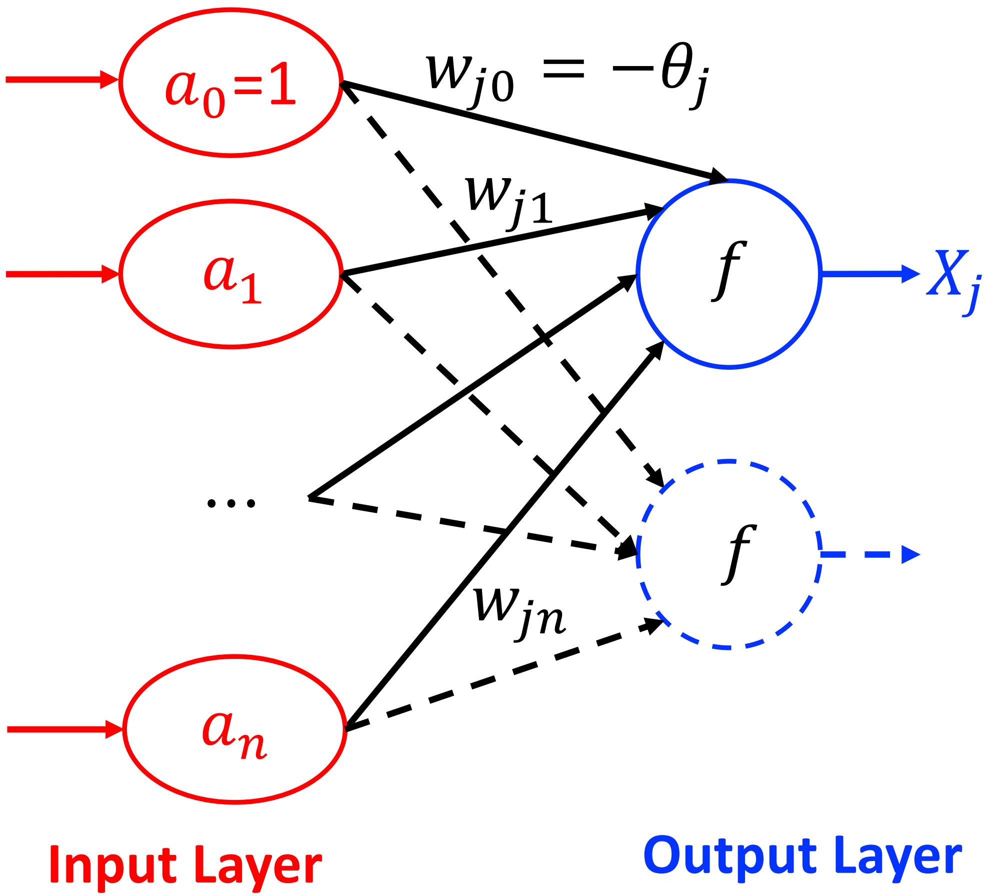
Perceptron Training



In training, the output neuron first computes an output as:

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

Perceptron Training

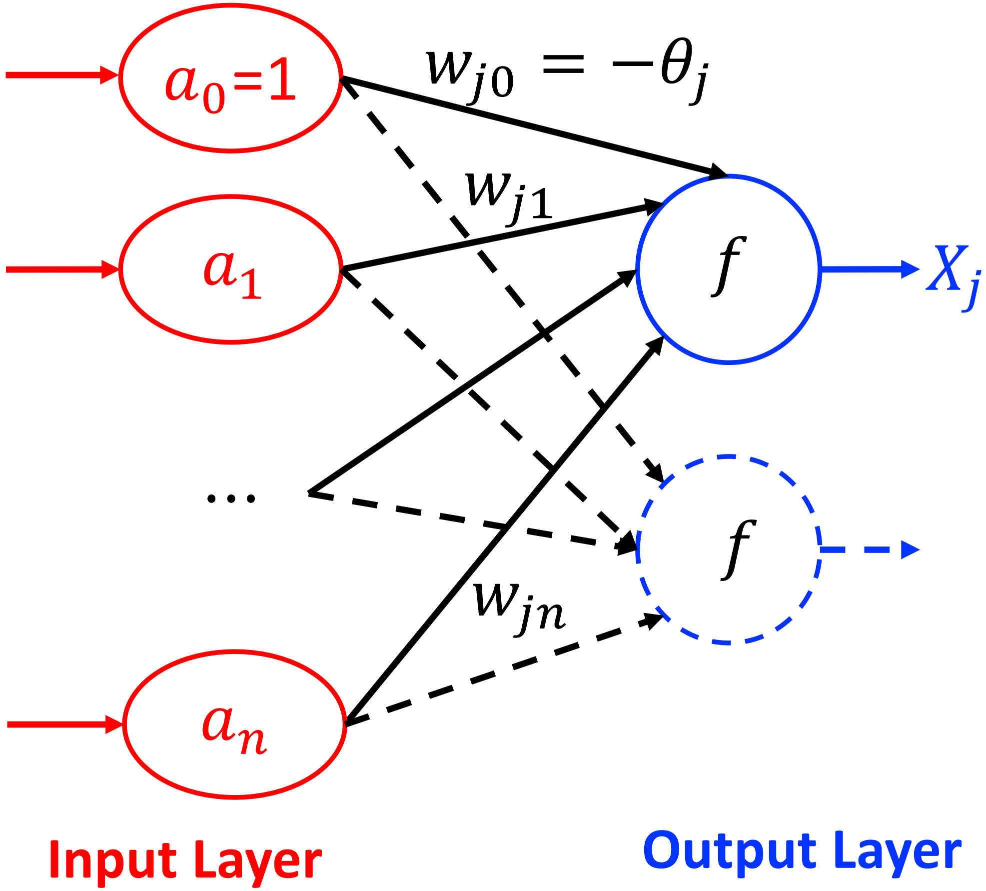


In training, the output neuron first computes an output as:

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

The network **outputs** $\boldsymbol{X_j}$ are then **compared to the desired outputs** specified in the training set and obtain the **error**.

Perceptron Training



In training, the output neuron first computes an output as:

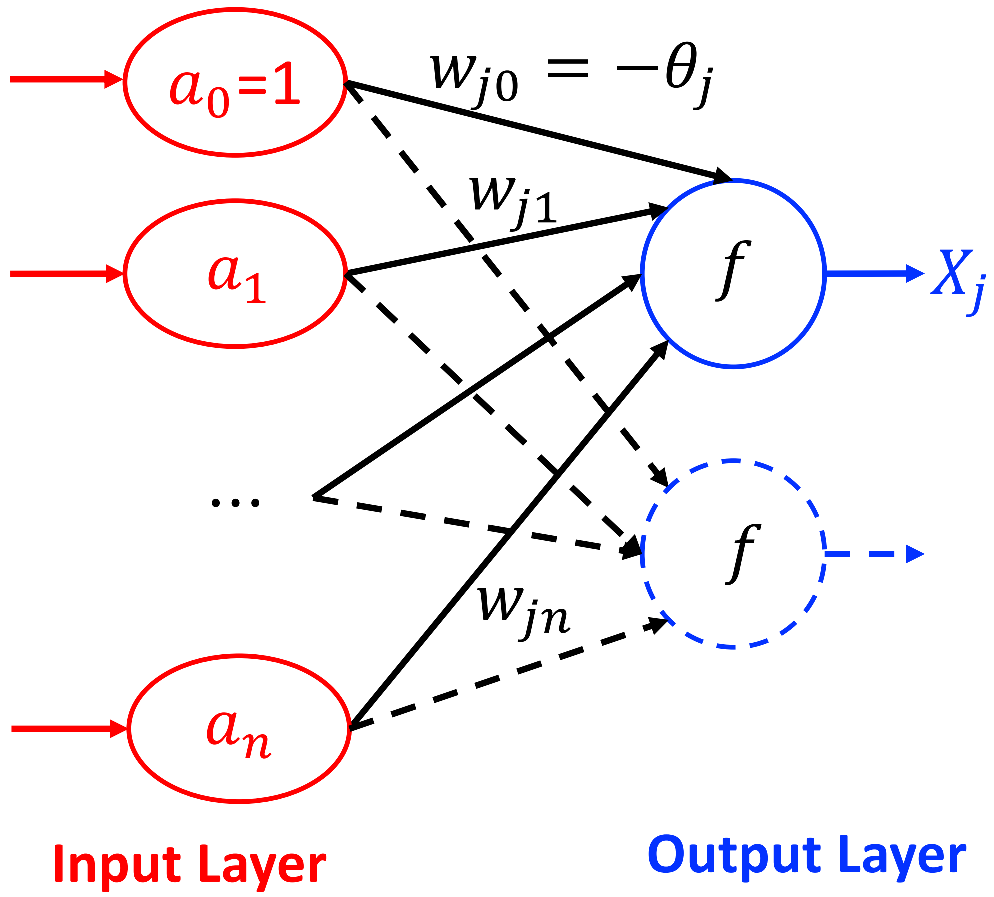
$$X_j = f(S_j) = \begin{cases} 1, & S_j \geq 0, \\ 0, & S_j < 0. \end{cases}$$

$$e_j = (t_j - X_j)$$

where t_j is the (binary) target.

The *error of an output neuron* is the difference between the target output and the instant one.

Perceptron Training



In training, the output neuron first computes an output as:

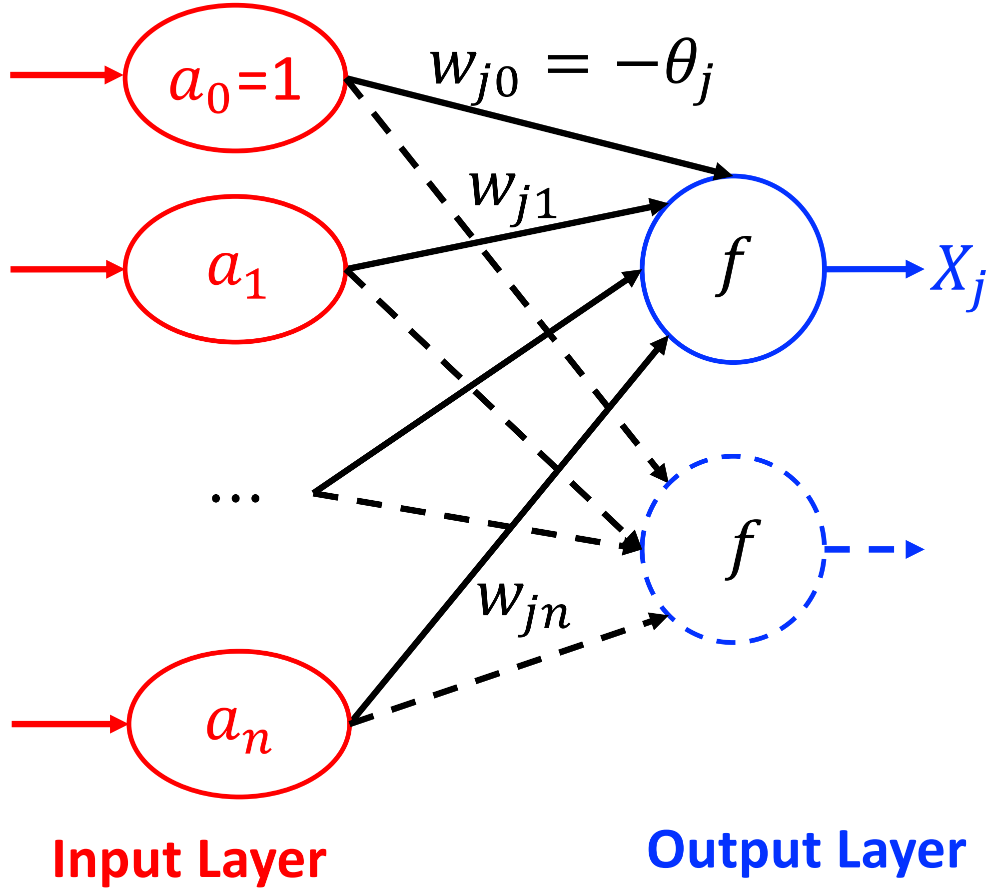
$$X_j = f(S_j) = \begin{cases} 1, & S_j \geq 0, \\ 0, & S_j < 0. \end{cases}$$

$$e_j = (t_j - X_j)$$

where t_j is the (binary) target.

The *errors* are then used to readjust the values of the weights of the connections.

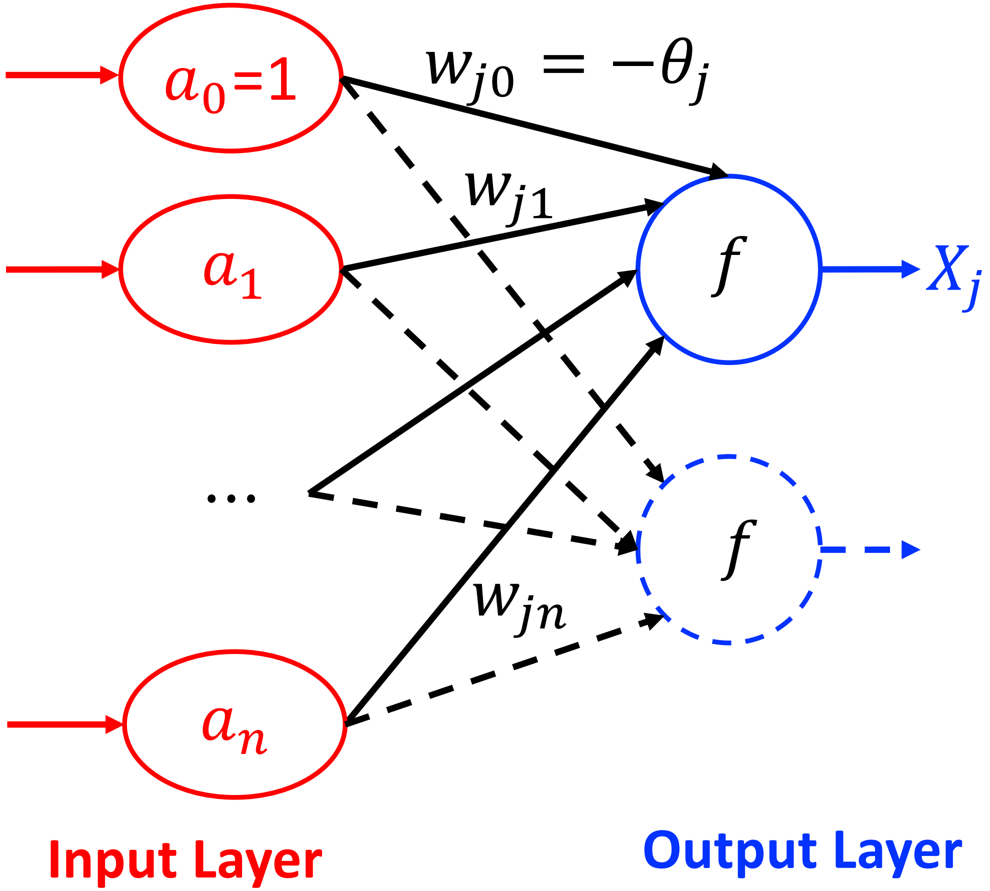
Perceptron Training



The weights readjustment is done in such a way that the network is – on the whole – more likely to give the desired response next time.

The goal of the training is to arrive at a single set of weights that allow each input in the training set to be mapped to the correct output by the network.

Weight Update



1. Compute "error" of very connection for every output neuron:

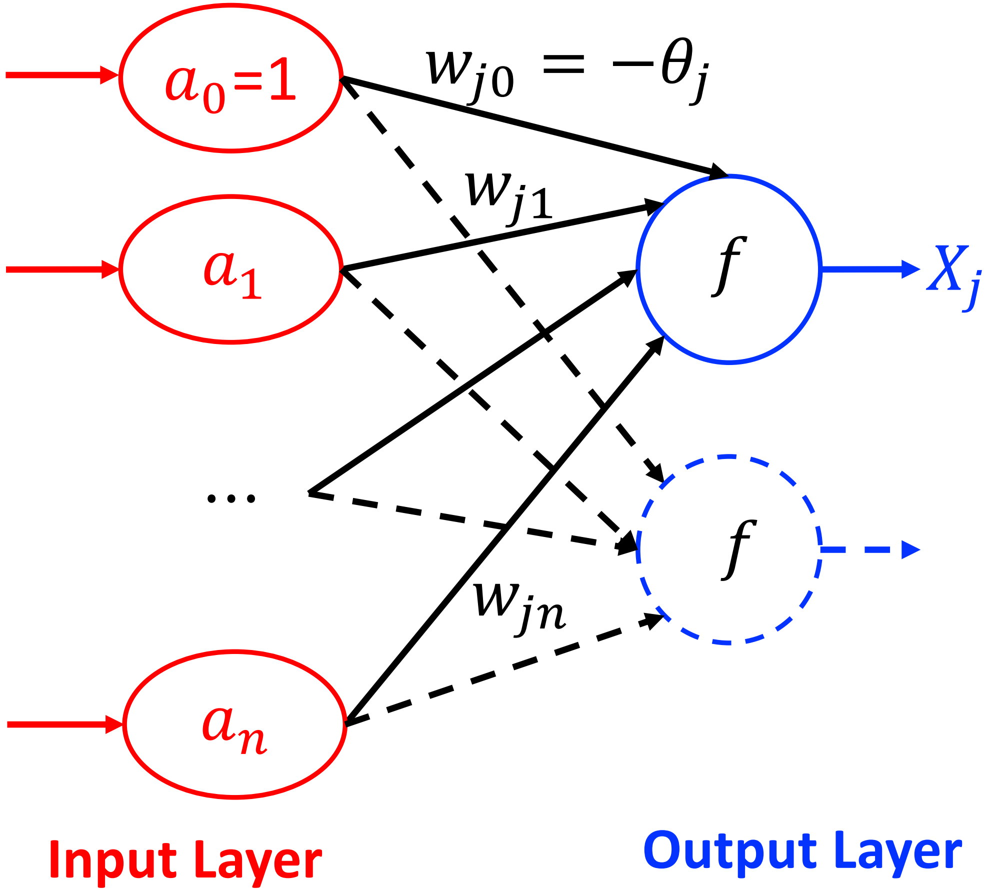
$$e_j^k = (t_j^k - X_j^k)$$

where

t_j^k: the target value for the j-th output neuron for the k-th input pattern in the data set,

X_j^k: the instant value for the j-th output neuron for the k-th input pattern.

Weight Update



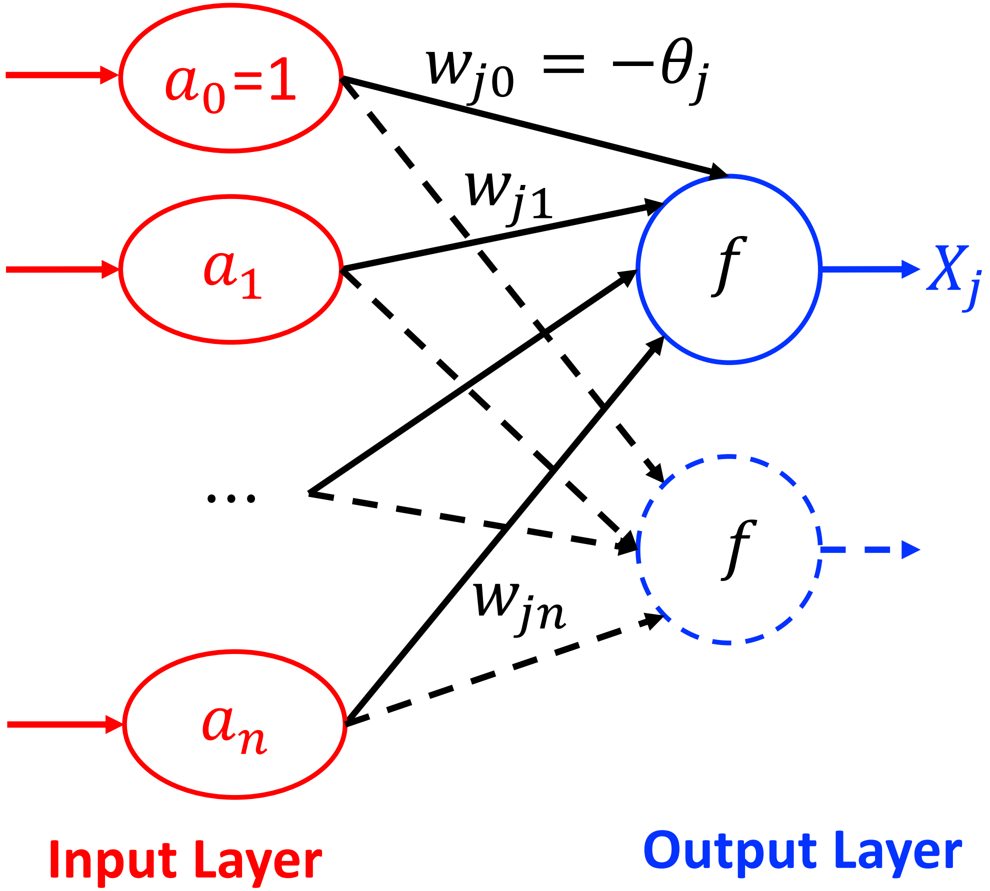
2. Update the weights:

$$w_{ji}^k = w_{ji}^k + \Delta w_{ji}^k$$

where

$$\Delta w_{ji}^k = C e_j^k a_i^k$$

Weight Update



2. Update the weights:

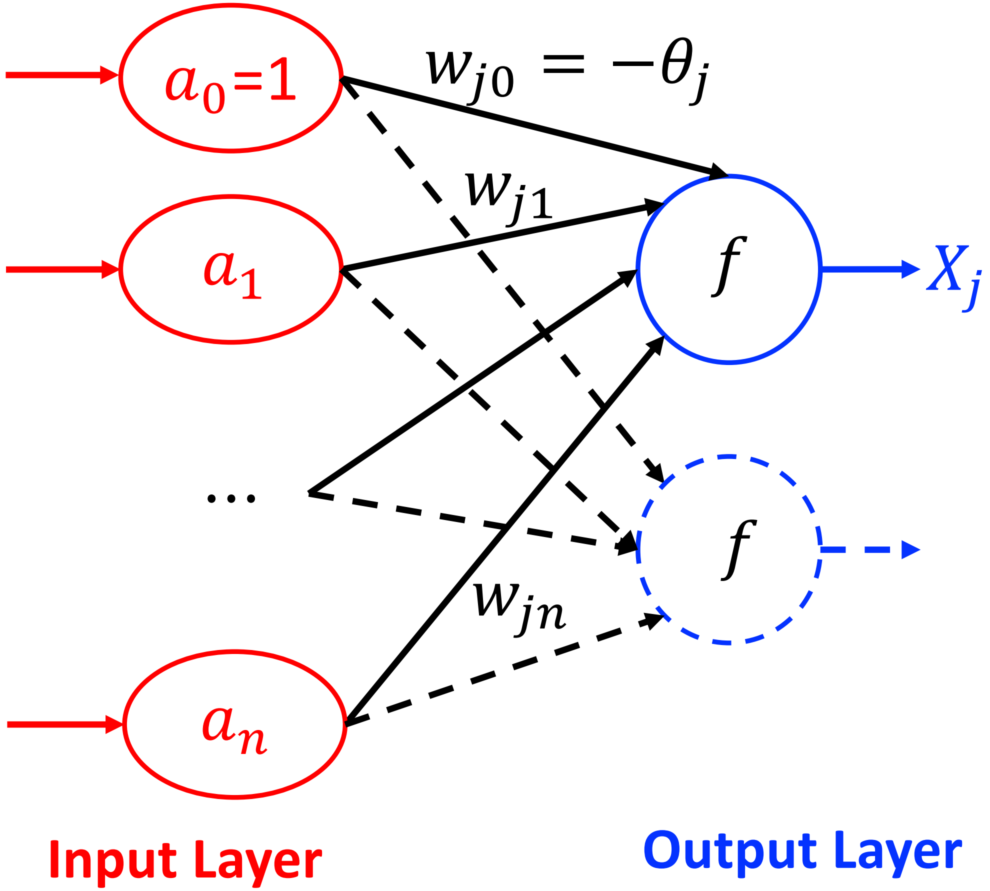
$$w_{ji}^k = w_{ji}^k + \Delta w_{ji}^k$$

where

$$\Delta w_{ji}^k = C e_j^k a_i^k$$

Perceptron Learning Rule!

Weight Update



2. Update the weights:

$$w_{ji}^k = w_{ji}^k + \Delta w_{ji}^k$$

where

$$\Delta w_{ji}^k = C e_j^k a_i^k$$

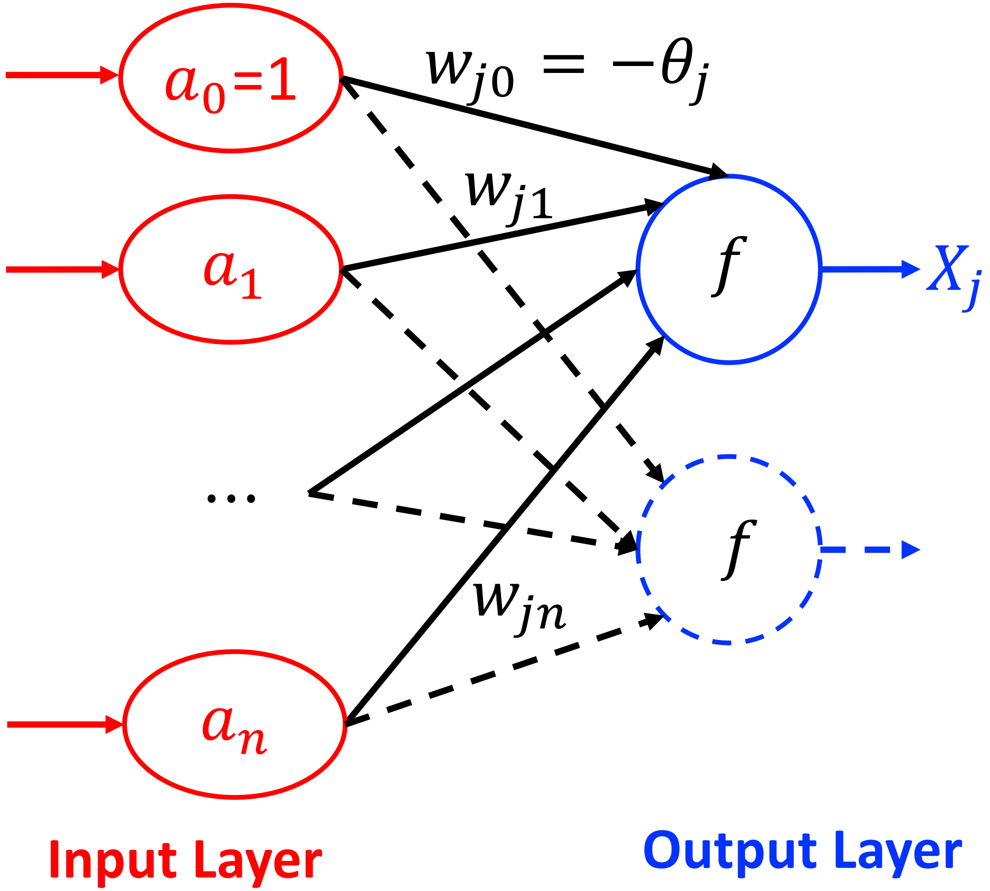
Perceptron Learning Rule!

According to the learning rule, a weight of connection changes

If and only if both the input value and the error of the output are not equal to 0.

Q: Difference with Hebb's rule?

Weight Update



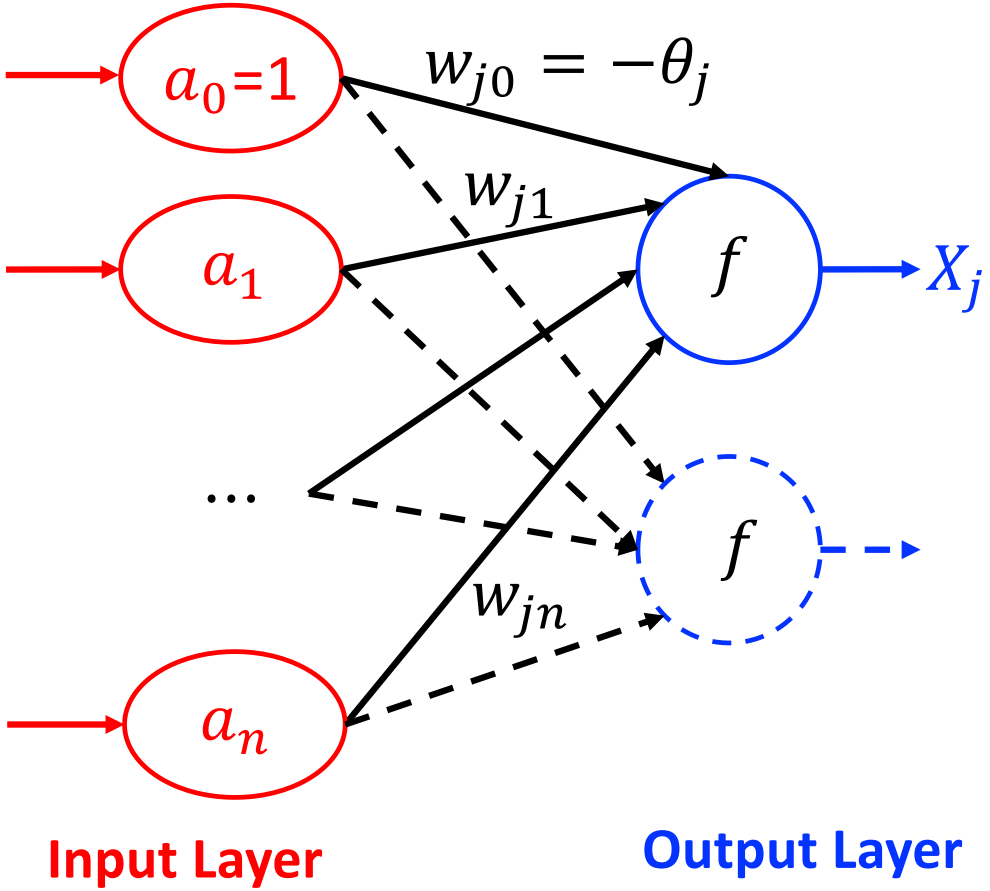
Perceptron Learning Rule :

$$\Delta w_{ji}^k = C e_j^k a_i^k$$

As outputs and target values are binary,

$$e_j^k = t_j^k - X_j^k = \begin{cases} 1, & t_j^k = 1, X_j^k = 0 \\ 0, & t_j^k = X_j^k \\ -1, & t_j^k = 0, X_j^k = 1 \end{cases}$$

Weight Update



Perceptron Learning Rule :

$$\Delta w_{ji}^k = C e_j^k a_i^k$$

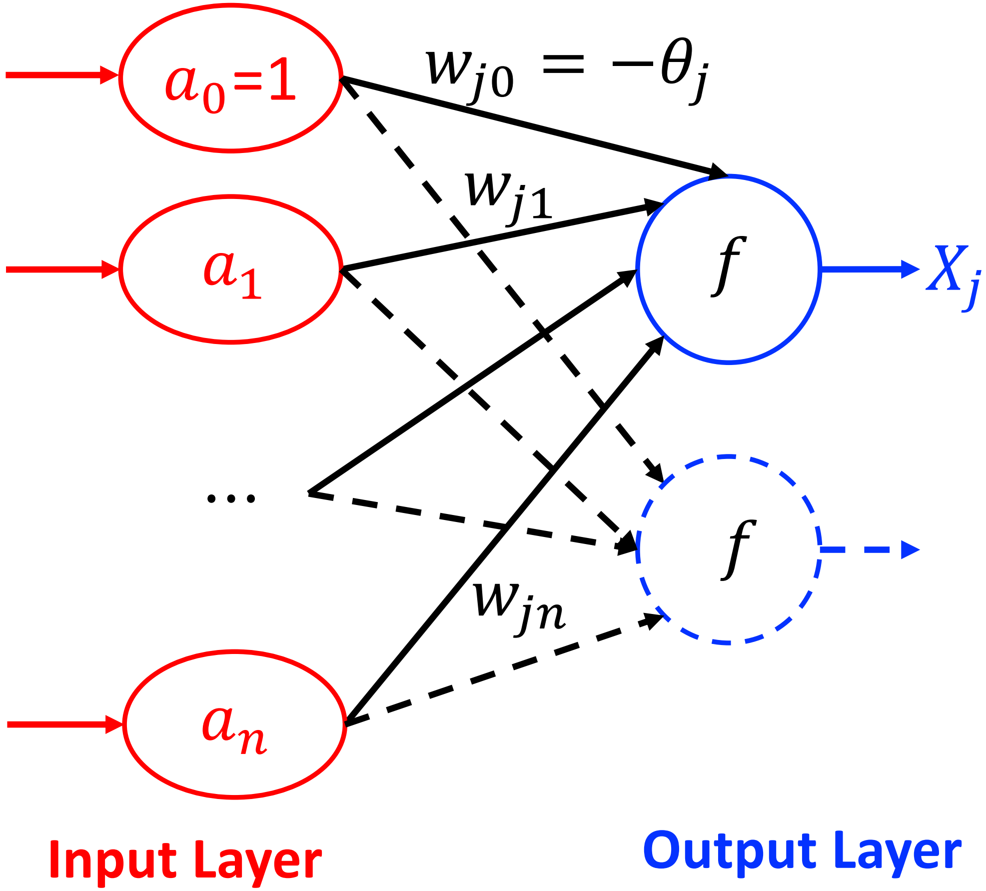
As outputs and target values are binary,

$$e_j^k = t_j^k - X_j^k = \begin{cases} 1, & t_j^k = 1, X_j^k = 0 \\ 0, & t_j^k = X_j^k \\ -1, & t_j^k = 0, X_j^k = 1 \end{cases}$$

The value of the "learning rate" C is

- usually set below 1 and
- ***determines the amount of correction made in a single iteration***.

Weight Update



Perceptron Learning Rule :

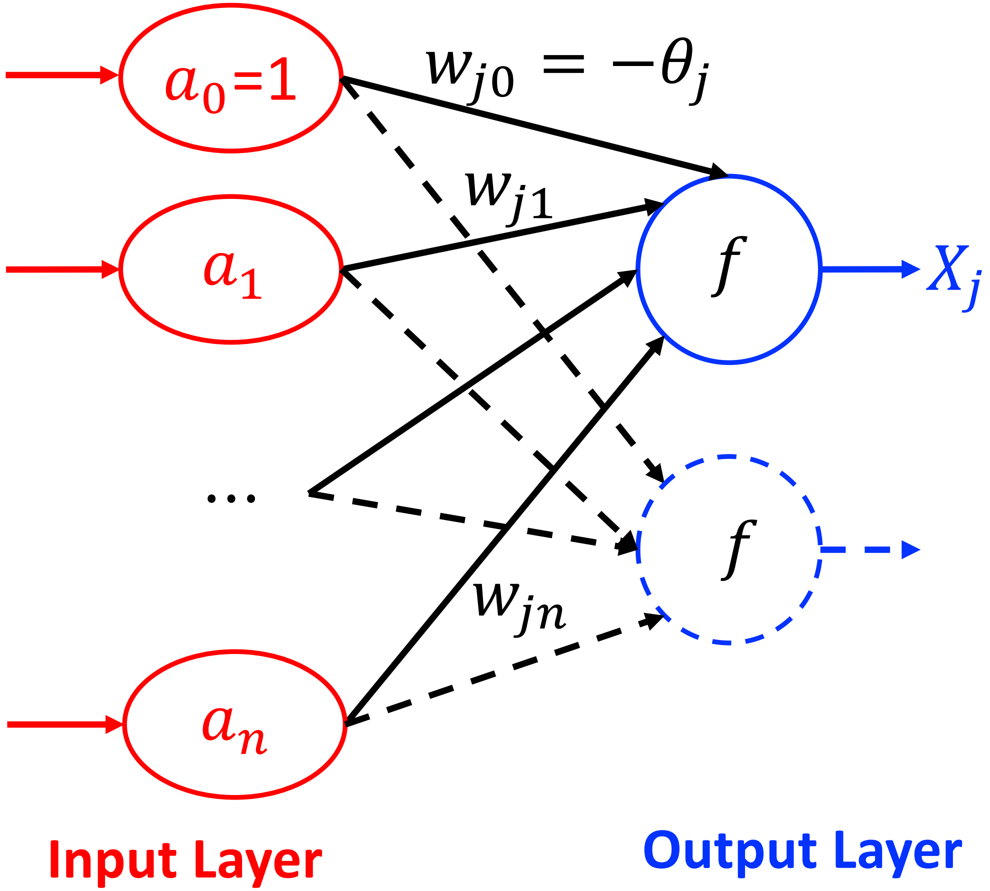
$$\Delta w_{ji}^k = C e_j^k a_i^k$$

As outputs and target values are binary,

$$e_j^k = t_j^k - X_j^k = \begin{cases} 1, & t_j^k = 1, X_j^k = 0 \\ 0, & t_j^k = X_j^k \\ -1, & t_j^k = 0, X_j^k = 1 \end{cases}$$

The ***overall learning time of the network*** is affected by the value of the "learning rate" C: slower for small values and faster for larger.

Network Performance

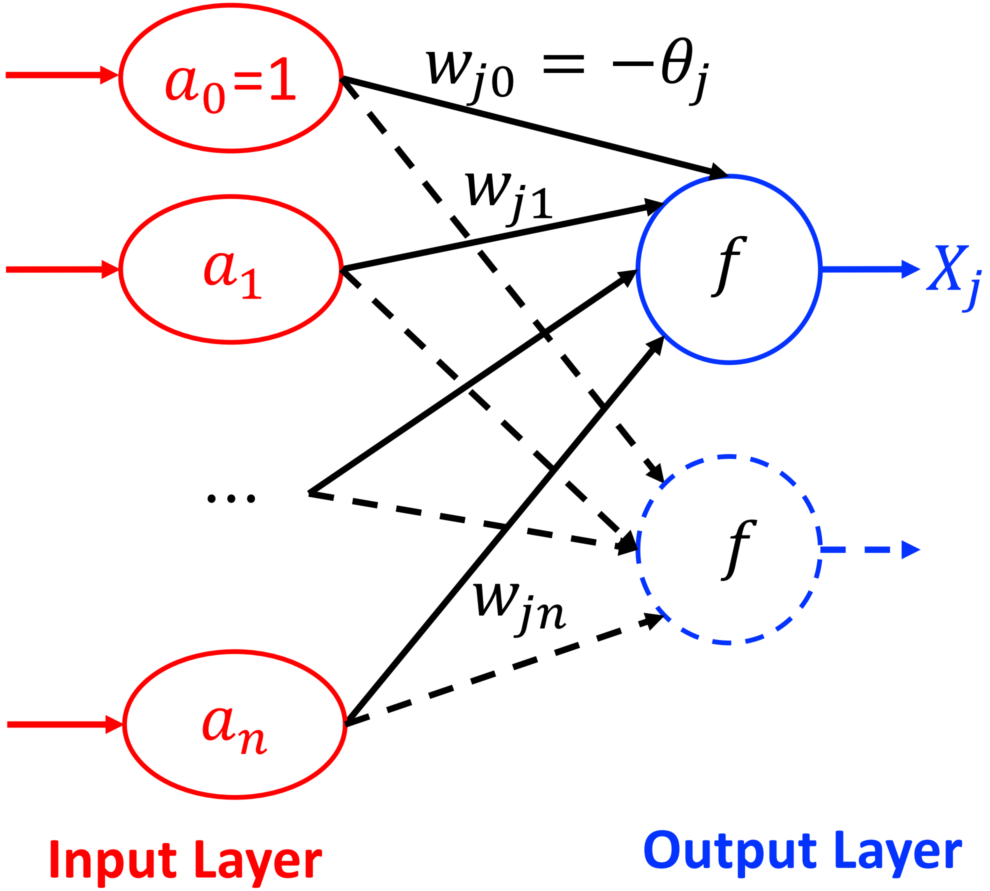


$$\text{RMS} = \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(e_j^k\right)^2}{rm}}$$
$$= \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(t_j^k - X_j^k\right)^2}{rm}}$$

where
r: the number of patterns in the data set,
m: the number of output neurons.

The ***network performance*** during training session is measured by a root-mean-square (RMS) error value.

Network Performance



$$\text{RMS} = \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(e_j^k\right)^2}{rm}}$$

$$= \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(t_j^k - X_j^k\right)^2}{rm}}$$

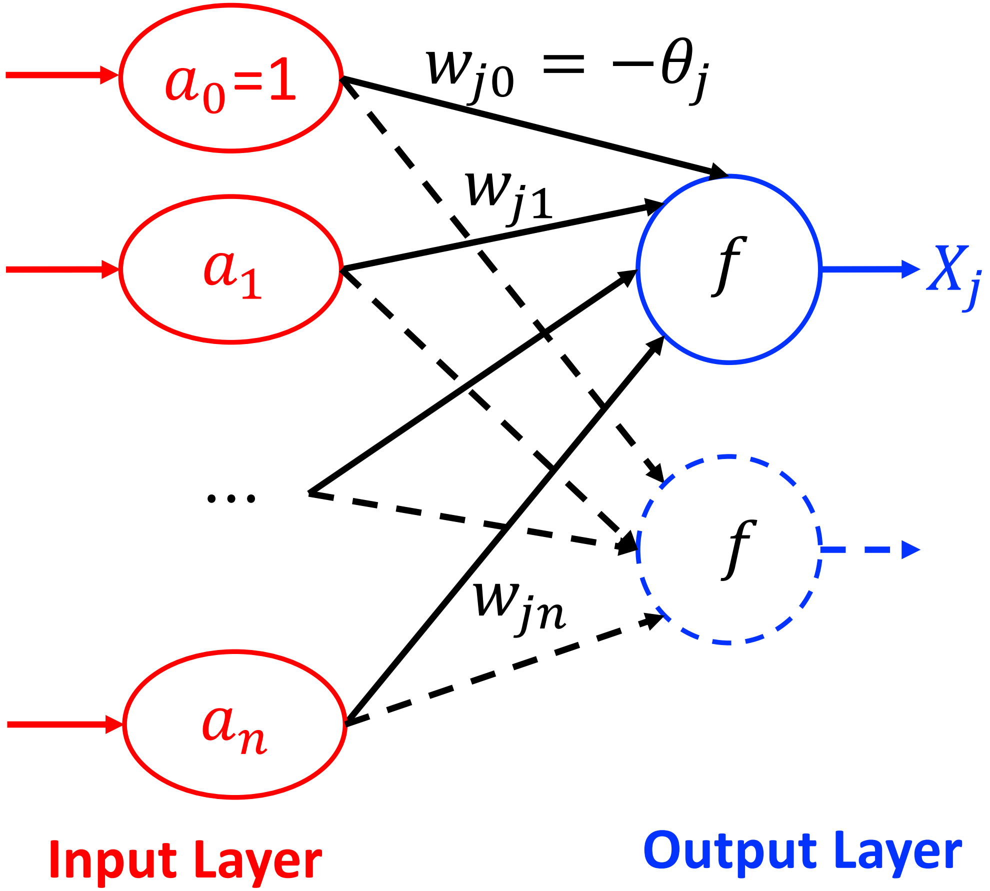
where

r: the number of patterns in the data set,

m: the number of output neurons.

When **RMS is closed to zero**, we stop the training.

Network Performance



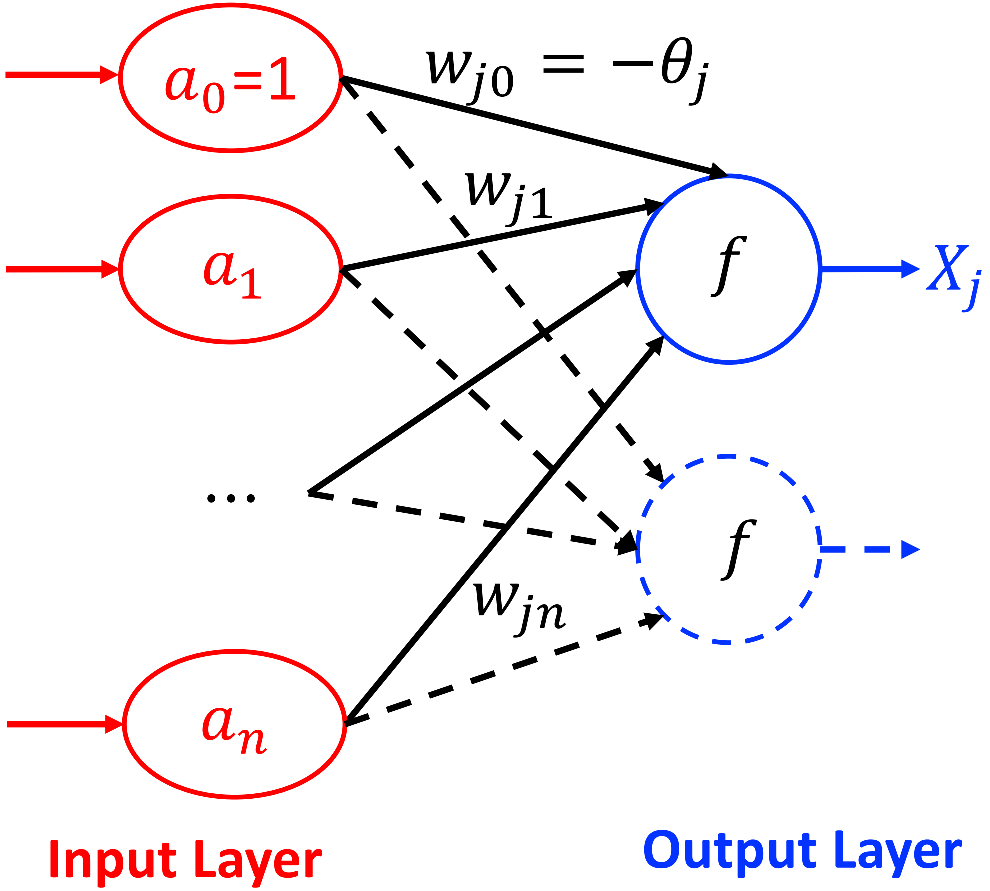
$$\text{RMS} = \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(e_j^k\right)^2}{rm}}$$

$$= \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(t_j^k - X_j^k\right)^2}{rm}}$$

where
r: the number of patterns in the data set,
m: the number of output neurons.

The first sum is taken over all patterns in the training set, and ***the second sum*** is taken over all output neurons.

Network Performance



$$\text{RMS} = \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(e_j^k\right)^2}{rm}}$$

$$= \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(t_j^k - X_j^k\right)^2}{rm}}$$

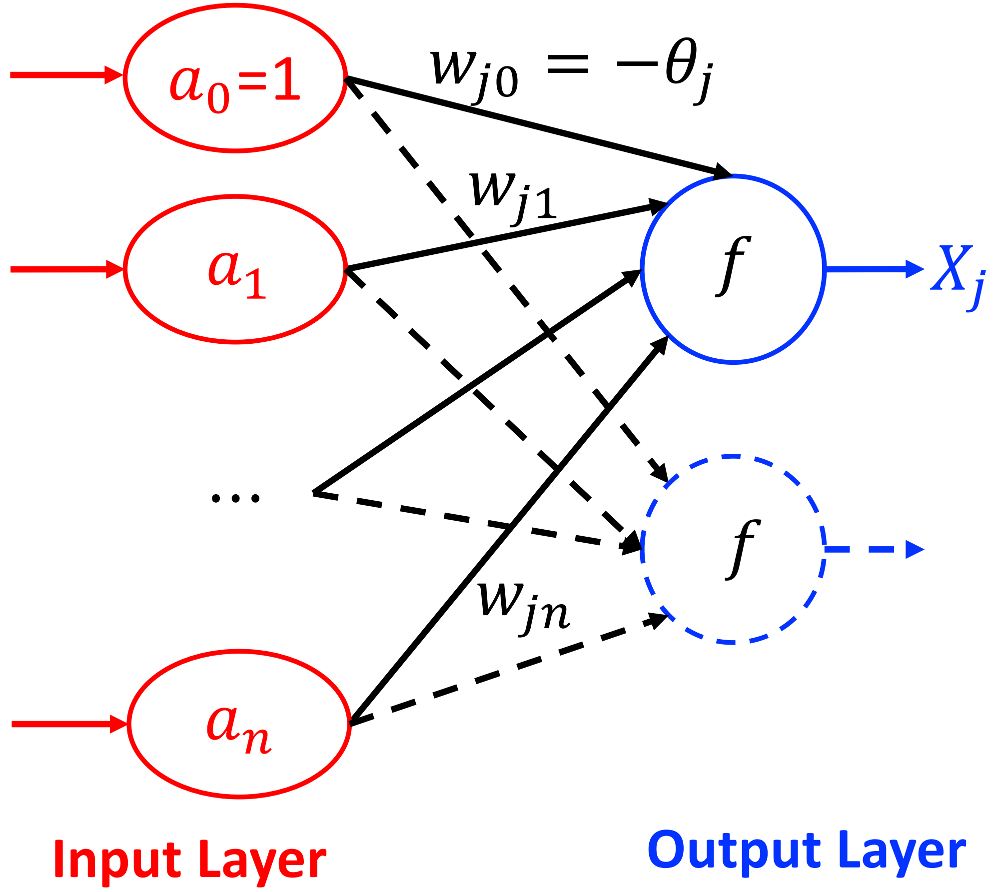
where

r: the number of patterns in the data set,

m: the number of output neurons.

As the target output values t_j^k and r and m numbers are constants, **the *RMS* error is a function of the instant outputs X_j^k only.**

Network Performance



$$\text{RMS} = \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(e_j^k\right)^2}{rm}}$$

$$= \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(t_j^k - X_j^k\right)^2}{rm}}$$

where

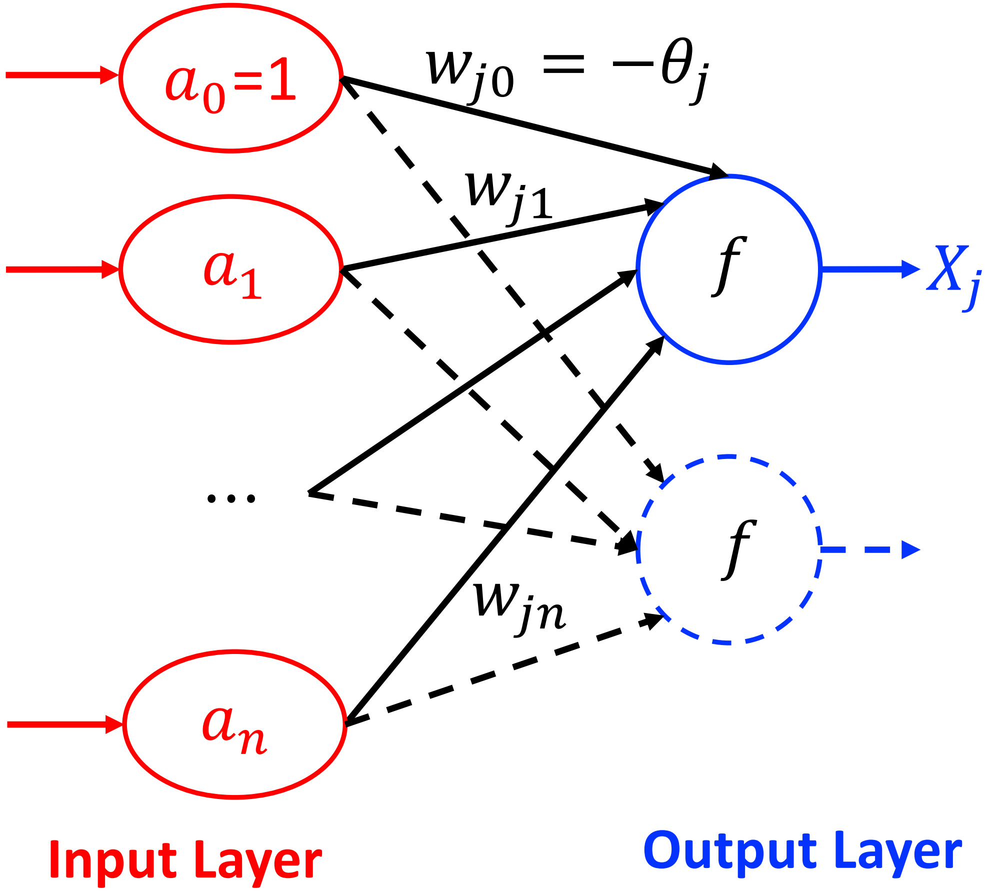
r: the number of patterns in the data set,

m: the number of output neurons.

In turn, **the instant outputs** X_j^k are also functions of the input values a_i^k, which also are constant, and the weights w_{ji}^k:

$$X_j^k = f\left(\sum_{i=0}^{n} w_{ji}^k a_i^k\right) = \bar{f}_{a^k}\left(w_{ji}^k\right)$$

Network Performance



$$\text{RMS} = \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(e_j^k\right)^2}{rm}}$$

$$= \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(t_j^k - X_j^k\right)^2}{rm}}$$

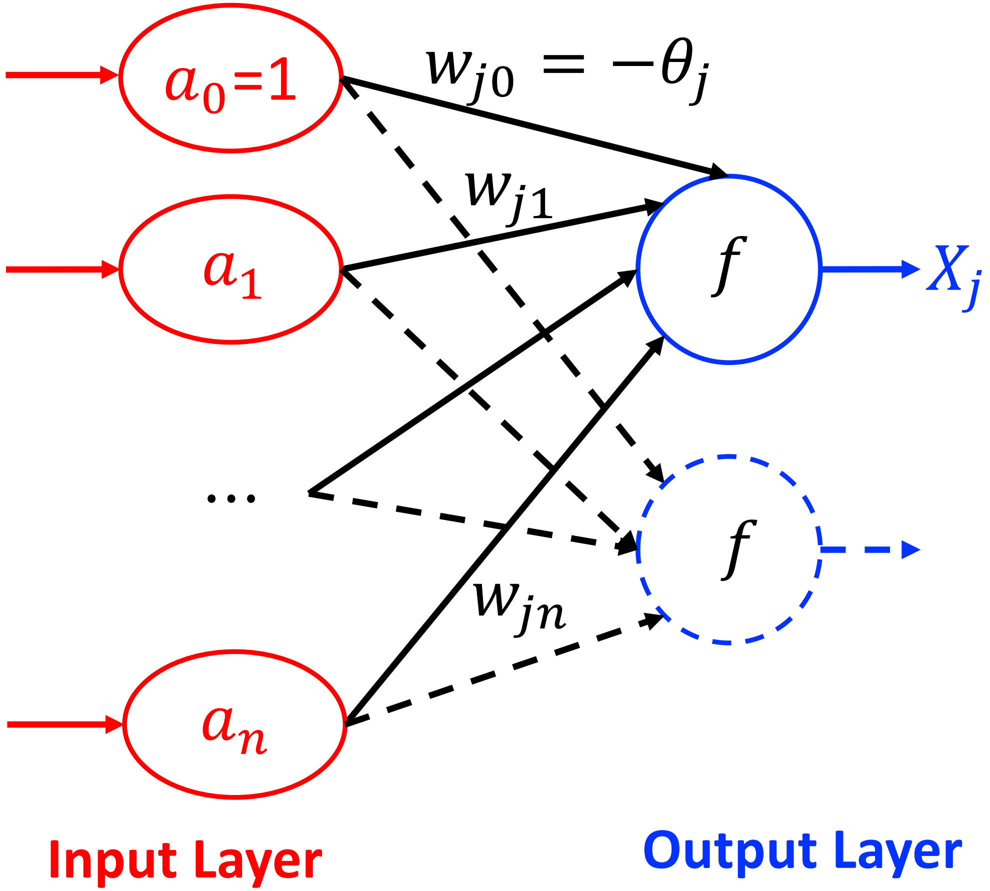
where
r: the number of patterns in the data set,
m: the number of output neurons.

Thus, RMS error can be represented as:

$$\text{RMS} = F_D(w)$$

where w is network weight, D is data set.

Network Performance



$$\mathrm{RMS} = \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(e_j^k\right)^2}{rm}} = \sqrt{\frac{\sum_{k=1}^{r}\sum_{j=1}^{m}\left(t_j^k - X_j^k\right)^2}{rm}}$$

where
r: the number of patterns in the data set,
m: the number of output neurons.

The best performance of the network over a given data set D corresponds to **the minimum of the RMS error**, and we adjust the weight of connections w in order to reach the minimum.

Perceptron Learning Algorithm

Algorithm 1: Perceptron Learning Algorithm

Data: Labelled data set D: r n-dimensional input points, each of which has m labels. Small positive real δ. Learning rate C.

Result: Weight matrix $w = [w_1, \cdots, w_m]$

Initialize weights w randomly;

while *!convergence (RMS* $\leq \delta$*)* **do**

  Pick random $a' \in D$;

  $a \longleftarrow [1, a']$;

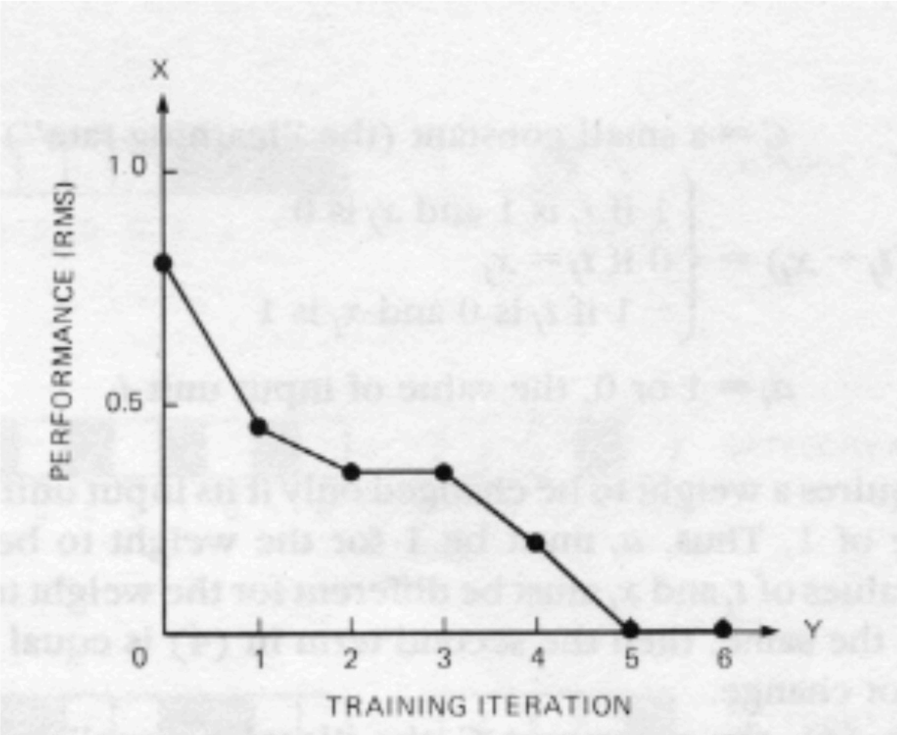
  for $j = 1, \cdots, m$ **do**

    `/* We represent the learning rule in the vector form */`

    $w_j = w_j + C(t_j - X_j)a$;

return w;

Network Performance



Thus, RMS error can be represented as:

$$\mathrm{RMS} = F_D(w)$$

where w is network weight, D is data set.

Learning curve: dependency of the RMS error on the number of iterations.

- ***Initially***, the adaptable ***weights are all set to small random values***, and the network does not perform very well;
- Performance improves **during training**;
- Finally, the error gets close to zero, training stops. We say the network has **converged**.