Artificial Neural Networks

The perceptron learning rule

Artificial neural systems, or neural networks, are physical cellular systems which can acquire, store, and utilize experiental knowledge

The knowledge is in the form of stable states or mappings embedded in networks that can be recalled in response to the presentation of cues.

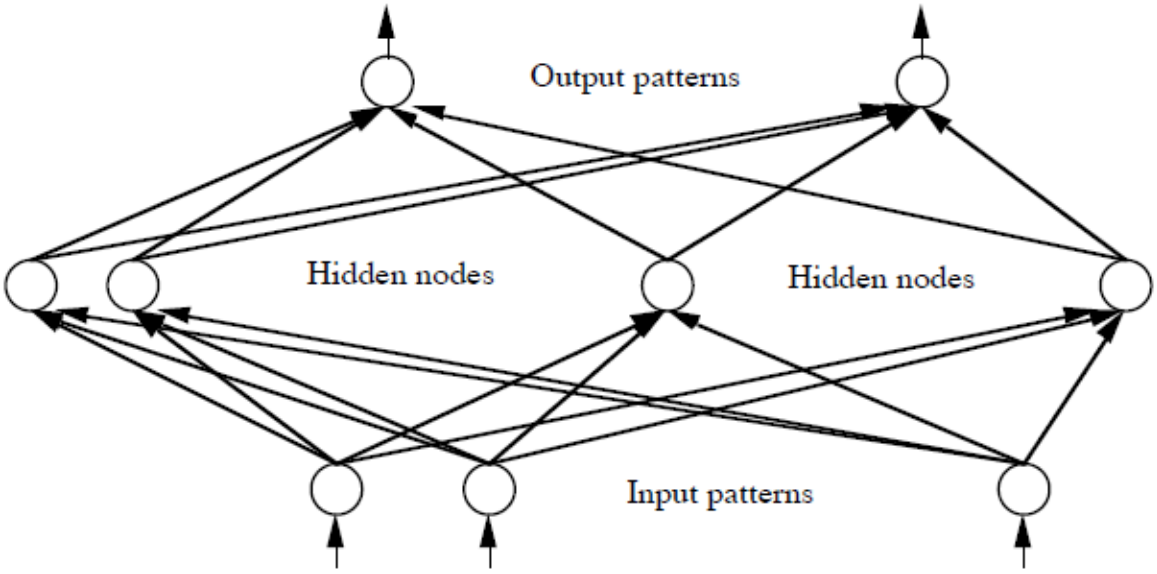


Figure 2.1 A multi-layer feedforward neural network.

The basic processing elements of neural networks are called *artificial neurons*, or simply *neurons* or *nodes*.

Each processing unit is characterized by an activity level (representing the state of polarization of a neuron), an output value (representing the firing rate of the neuron), a set of input connections, (representing synapses on the cell and its dendrite), a bias value (representing an internal resting level of the neuron), and a set of output connections (representing a neuron's axonal projections). Each of these aspects of the unit are represented mathematically by real numbers. Thus, each connection has an associated weight (synaptic strength) which determines the effect of the incoming input on the activation level of the unit. The weights may be positive (excitatory) or negative (inhibitory).

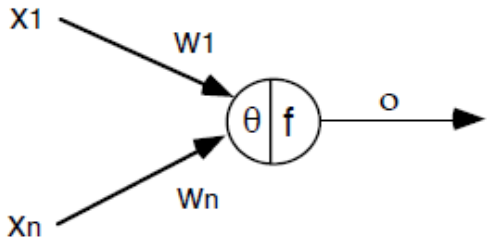


Figure 2.1a A processing element with single output connection.

The signal flow from of neuron inputs, x_j, is considered to be unidirectionalas indicated by arrows, as is a neuron's output signal flow. The neuron output signal is given by the following relationship

$$o = f(< w, x >) = f(w^T x) = f(\sum_{j=1}^{n} w_j x_j)$$

where $w = (w_1, \ldots, w_n)^T \in I\!R^n$ is the weight vector. The function $f(w^T x)$ is often referred to as an *activation (or transfer) function.* Its domain is the set of activation values, *net*, of the neuron model, we thus often use this function as $f(net)$. The variable *net* is defined as a scalar product of the weight and input vectors

$$net =< w, x >= w^T x = w_1 x_1 + \cdots + w_n x_n$$

and in the simplest case the output value o is computed as

$$o = f(net) = \begin{cases} 1 & \text{if } w^T x \geq \theta \\ 0 & \text{otherwise,} \end{cases}$$

where θ is called threshold-level and this type of node is called a *linear threshold unit.*

Example 2.1.1 *Suppose we have two Boolean inputs $x_1, x_2 \in \{0, 1\}$, one Boolean output $o \in \{0, 1\}$ and the training set is given by the following input/output pairs*

	x_1	x_2	$o(x_1, x_2) = x_1 \wedge x_2$
1.	*1*	*1*	*1*
2.	*1*	*0*	*0*
3.	*0*	*1*	*0*
4.	*0*	*0*	*0*

Then the learning problem is to find weight w_1 and w_2 and threshold (or bias) value θ such that the computed output of our network (which is given by the linear threshold function) is equal to the desired output for all examples. A straightforward solution is $w_1 = w_2 = 1/2$, $\theta = 0.6$. Really, from the equation

$$o(x_1, x_2) = \begin{cases} 1 & \text{if } x_1/2 + x_2/2 \geq 0.6 \\ 0 & \text{otherwise} \end{cases}$$

it follows that the output neuron fires if and only if both inputs are on.

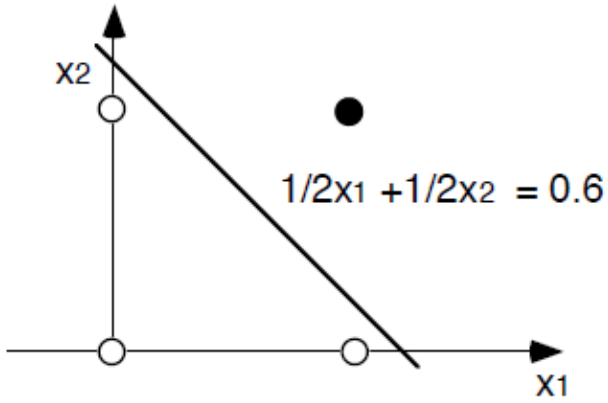


Figure 2.2 *A solution to the learning problem of Boolean* and *function.*

Example 2.1.2 *Suppose we have two Boolean inputs $x_1, x_2 \in \{0, 1\}$, one Boolean output $o \in \{0, 1\}$ and the training set is given by the following input/output pairs*

	x_1	x_2	$o(x_1, x_2) = x_1 \vee x_2$
1.	*1*	*1*	*1*
2.	*1*	*0*	*1*
3.	*0*	*1*	*1*
4.	*0*	*0*	*0*

Then the learning problem is to find weight w_1 and w_2 and threshold value θ such that the computed output of our network is equal to the desired output for all examples. A straightforward solution is $w_1 = w_2 = 1$, $\theta = 0.8$. Really, from the equation

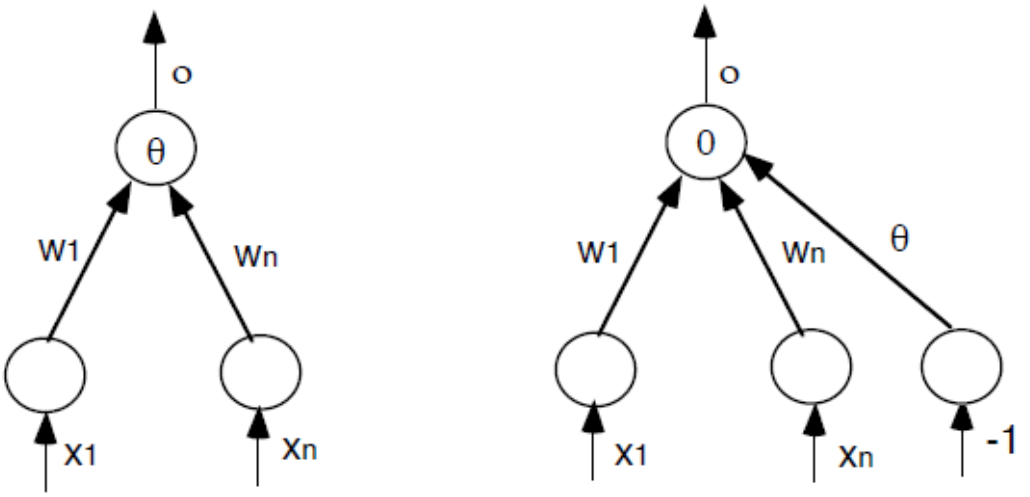
$$o(x_1, x_2) = \begin{cases} 1 & \text{if } x_1 + x_2 \geq 0.8 \\ 0 & \text{otherwise} \end{cases}$$

it follows that the output neuron fires if and only if at least one of the inputs is on.

The removal of the threshold from our network is very easy by increasing the dimension of input patterns. Really, the identity

$$w_1x_1 + \cdots + w_nx_n > \theta \iff w_1x_1 + \cdots + w_nx_n - 1 \times \theta > 0$$

means that by adding an extra neuron to the input layer with fixed input value -1 and weight θ the value of the threshold becomes zero. It is why in the following we suppose that the thresholds are always equal to zero.



We define now the scalar product of n-dimensional vectors, which plays a very important role in the theory of neural networks.

Definition 2.1.2 *Let $w = (x_n, \ldots, w_n)^T$ and $x = (x_1, \ldots, x_n)^T$ be two vectors from $\mathbb{R}^n$. The scalar (or inner) product of w and x, denoted by $< w, x >$ or $w^T x$, is defined by*

$$< w, x >= w_1x_1 + \cdots + w_nx_n = \sum_{j=1}^{n} w_jx_j$$

Other definition of scalar product in two dimensional case is

$$< w, x >= \|w\|\|x\| \cos(w, x)$$

where $\|.\|$ denotes the Eucledean norm in the real plane, i.e.

$$\|w\| = \sqrt{w_1^2 + w_2^2},\ \|x\| = \sqrt{x_1^2 + x_2^2}$$

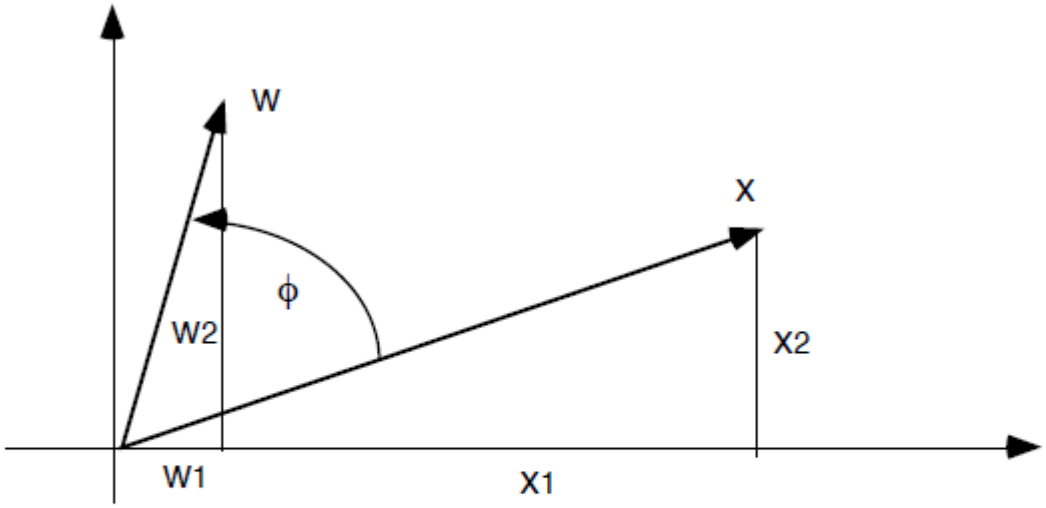


Figure 2.4 $w = (w_1, w_2)^T$ and $x = (x_1, x_2)^T$.

Lemma 2.1.1 *The following property holds*

$$< w, x >= w_1 x_1 + w_2 x_2 = \sqrt{w_1^2 + w_2^2}\sqrt{x_1^2 + x_2^2}\cos(w, x) = \|w\|\|x\|cos(w, x)$$

The perceptron learning rule, introduced by Rosenblatt [21], is a typical error correction learning algorithm of single-layer feedforward networks with linear threshold activation function.

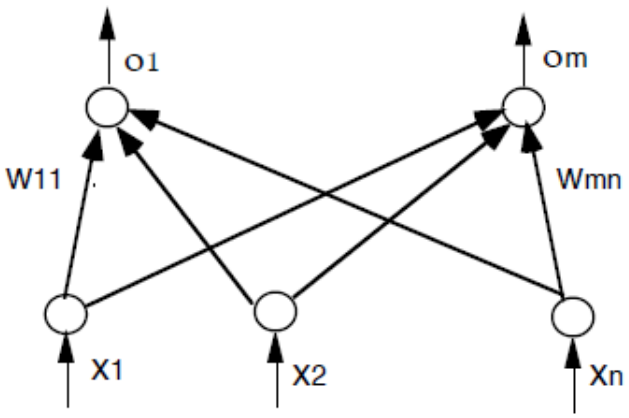


Figure 2.6 Single-layer feedforward network.

Usually, w_{ij} denotes the weight from the j-th input unit to the i-th output unit and w_i denotes the weight vector of the i-th output node.

We are given a training set of input/output pairs

No.	input values	desired output values
1.	$x^1 = (x_1^1, \ldots x_n^1)$	$y^1 = (y_1^1, \ldots, y_m^1)$
$\vdots$	$\vdots$	$\vdots$
K.	$x^K = (x_1^K, \ldots x_n^K)$	$y^K = (y_1^K, \ldots, y_m^K)$

Our problem is to find weight vectors w_i such that

$$o_i(x^k) = \text{sign}(< w_i, x^k >) = y_i^k, \ i = 1, \ldots, m$$

for all training patterns k.

The activation function of the output nodes is linear threshold function of the form

$$o_i(x) = \text{sign}(< w_i, x >) = \begin{cases} +1 & \text{if } < w_i, x > \geq 0 \\ -1 & \text{if } < w_i, x > < 0 \end{cases}$$

and the weight adjustments in the perceptron learning method are performed by

$$w_i := w_i + \eta(y_i^k - \text{sign}(< w_i, x^k >))x^k, \ i = 1, \ldots, m$$

$$w_{ij} := w_{ij} + \eta(y_i^k - \text{sign}(< w_i, x^k >))x_j^k, \ j = 1, \ldots, n$$

where $\eta > 0$ is the learning rate.

From this equation it follows that if the desired output is equal to the computed output, $y_i^k = \text{sign}(< w_i, x^k >)$, then the weight vector of the i-th output node remains unchanged, i.e. w_i is adjusted if and only if the computed output, $o_i(x^k)$, is incorrect. The learning stops when all the weight vectors remain unchanged during a complete training cycle.

Consider now a single-layer network with one output node. Then the input components of the training patterns can be classified into two disjunct classes

$$C_1 = \{x^k | y^k = 1\}, \quad C_2 = \{x^k | y^k = -1\}$$

i.e. x belongs to class C_1 if there exists an input/output pair $(x, 1)$, and x belongs to class C_2 if there exists an input/output pair $(x, -1)$.
Taking into consideration the definition of the activation function it is easy to see that we are searching for a weight vector w such that

$$< w, x > \geq 0 \text{ for each } x \in C_1, \text{ and } < w, x > < 0 \text{ for each } x \in C_2.$$

If such vector exists then the problem is called *linearly separable.*

Summary 2.1.1 *Perceptron learning algorithm.*

Given are K training pairs arranged in the training set

$$(x^1, y^1), \ldots, (x^K, y^K)$$

where $x^k = (x_1^k, \ldots, x_n^k)$, $y^k = (y_1^k, \ldots, y_m^k)$, $k = 1, \ldots, K$.

- **Step 1** $\eta > 0$ is chosen
- **Step 2** Weigts w_i are initialized at small random values, the running error E is set to 0, $k := 1$

- **Step 3** Training starts here. x^k is presented, $x := x^k$, $y := y^k$ and output o is computed

$$o_i(x) = \text{sign}(< w_i, x >), \ i = 1, \ldots, m$$

- **Step 4** Weights are updated

$$w_i := w_i + \eta(y_i - \text{sign}(< w_i, x >))x, \ i = 1, \ldots, m$$

- **Step 5** Cumulative cycle error is computed by adding the present error to E

$$E := E + \frac{1}{2}\|y - o\|^2$$

- **Step 6** If $k < K$ then $k := k+1$ and we continue the training by going back to **Step 3**, otherwise we go to **Step 7**
- **Step 7** The training cycle is completed. For $E = 0$ terminate the training session. If $E > 0$ then E is set to 0, $k := 1$ and we initiate a new training cycle by going to **Step 3**

Example 2.1.3 *Illustration of the perceptron learning algorithm.*

Consider the following training set

No.	input values	desired output value
1.	$x^1 = (1, 0, 1)^T$	-1
2.	$x^2 = (0, -1, -1)^T$	1
3.	$x^3 = (-1, -0.5, -1)^T$	1

The learning constant is assumed to be 0.1. The initial weight vector is $w^0 = (1, -1, 0)^T$.

Then the learning according to the perceptron learning rule progresses as follows.

Step 1 Input x^1, desired output is -1:

$$< w^0, x^1 >= (1, -1, 0) \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = 1$$

Correction in this step is needed since $y_1 = -1 \neq \text{sign}(1)$. We thus obtain the updated vector

$$w^1 = w^0 + 0.1(-1 - 1)x^1$$

Plugging in numerical values we obtain

$$w^1 = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} - 0.2 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.8 \\ -1 \\ -0.2 \end{pmatrix}$$

Step 2 Input is x^2, desired output is 1. For the present w^1 we compute the activation value

$$< w^1, x^2 >= (0.8, -1, -0.2) \begin{pmatrix} 0 \\ -1 \\ -1 \end{pmatrix} = 1.2$$

Correction is not performed in this step since $1 = \text{sign}(1.2)$, so we let $w^2 := w^1$.

Step 3 Input is x_3, desired output is 1.

$$< w^2, x^3 >= (0.8, -1, -0.2) \begin{pmatrix} -1 \\ -0.5 \\ -1 \end{pmatrix} = -0.1$$

Correction in this step is needed since $y_3 = 1 \neq \text{sign}(-0.1)$. We thus obtain the updated vector

$$w^3 = w^2 + 0.1(1+1)x^3$$

Plugging in numerical values we obtain

$$w^3 = \begin{pmatrix} 0.8 \\ -1 \\ -0.2 \end{pmatrix} + 0.2 \begin{pmatrix} -1 \\ -0.5 \\ -1 \end{pmatrix} = \begin{pmatrix} 0.6 \\ -1.1 \\ -0.4 \end{pmatrix}$$

Step 4 Input x^1, desired output is -1:

$$< w^3, x^1 >= (0.6, -1.1, -0.4) \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = 0.2$$

Correction in this step is needed since $y_1 = -1 \neq \text{sign}(0.2)$. We thus obtain the updated vector

$$w^4 = w^3 + 0.1(-1-1)x^1$$

Plugging in numerical values we obtain

$$w^4 = \begin{pmatrix} 0.6 \\ -1.1 \\ -0.4 \end{pmatrix} - 0.2 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.4 \\ -1.1 \\ -0.6 \end{pmatrix}$$

Step 5 Input is x^2, desired output is 1. For the present w^4 we compute the activation value

$$< w^4, x^2 >= (0.4, -1.1, -0.6) \begin{pmatrix} 0 \\ -1 \\ -1 \end{pmatrix} = 1.7$$

Correction is not performed in this step since $1 = \text{sign}(1.7)$, so we let $w^5 := w^4$.

Step 6 Input is x_3, desired output is 1.

$$< w^5, x^3 >= (0.4, -1.1, -0.6) \begin{pmatrix} -1 \\ -0.5 \\ -1 \end{pmatrix} = 0.75$$

Correction is not performed in this step since $1 = \text{sign}(0.75)$, so we let $w^6 := w^5$.

This terminates the learning process, because

$$< w^6, x_1 >= -0.2 < 0, \ < w^6, x_2 >= 1.7 > 0, \ < w^6, x_3 >= 0.75 > 0$$