Institute for Advanced Management Systems Research
Department of Information Technologies
Åbo Akademi University

The Perception Learning Rule - Tutorial

Robert Fullér

Directory

November 4, 2010
rfuller@abo.fi

Table of Contents

1. Artificial neural networks

Artificial neural systems can be considered as simplified mathematical models of brain-like systems and they function as parallel distributed computing networks.

However, in contrast to conventional computers, which are programmed to perform specific task, most neural networks must be taught, or trained.

They can learn new associations, new functional dependencies and new patterns. Although computers outperform both biological and artificial neural systems for tasks based on precise and fast arithmetic operations, artificial neural systems represent the promising new generation of information processing networks. The study of brain-style computation has its roots over 50 years ago in the work of McCulloch and Pitts (1943) and slightly later in Hebb's famous *Organization of Behavior* (1949).

The early work in artificial intelligence was torn between those who believed that intelligent systems could best be built on computers modeled after brains, and those like Minsky and Papert (1969) who believed that intelligence was fundamentally symbol processing of the kind readily modeled on the *von Neumann* computer.

For a variety of reasons, the symbol-processing approach became the dominant theme in *Artifcial Intelligence* in the 1970s.

However, the 1980s showed a rebirth in interest in neural computing:

1982 Hopfield provided the mathematical foundation for understanding the dynamics of an important class of networks.

1984 Kohonen developed unsupervised learning networks for feature mapping into regular arrays of neurons.

1986 Rumelhart and McClelland introduced the backpropagation learning algorithm for complex, multilayer networks.

Beginning in 1986-87, many neural networks research programs were initiated. The list of applications that can be solved by neural networks has expanded from small test-size examples to large practical tasks. Very-large-scale integrated neural network chips have been fabricated.

In the long term, we could expect that artificial neural systems will be used in applications involving vision, speech, decision making, and reasoning, but also as signal processors such as filters, detectors, and quality control systems.

Definition 1.1. *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.

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 con-

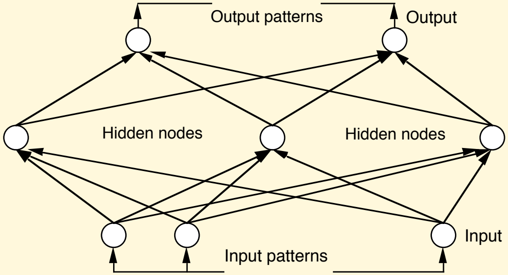

Figure 1: A multi-layer feedforward neural network.

nections, (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).

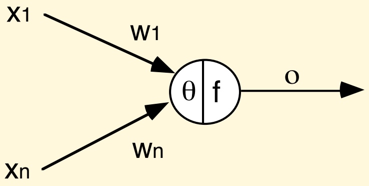

Figure 2: A processing element with single output connection.

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,

$$o = f(\langle w, x\rangle) = f(w^T x) = f\left(\sum_{j=1}^{n} w_j x_j\right)$$

where $w = (w_1, \dots, w_n)^T \in \mathbb{R}^n$ 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

$$net = \langle w, x\rangle = w^T x = w_1 x_1 + \cdots + w_n x_n$$

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 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.

Example 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*

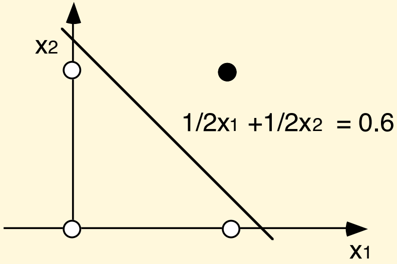

Figure 3: A solution to the learning problem of Boolean *and*.

	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,

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

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

$$w_1x_1 + \cdots + w_nx_n > \theta \iff$$

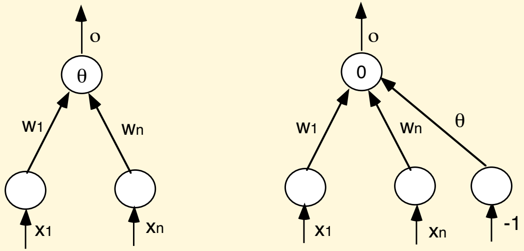

Figure 4: Removing the threshold.

$$w_1 x_1 + \cdots + w_n x_n - 1 \times \theta > 0$$

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 1.2. *Let $w = (w_1, \dots, w_n)^T$ and $x = (x_1, \dots, x_n)^T$ 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*

$$\langle w, x \rangle = w_1 x_1 + \cdots + w_n x_n = \sum_{j=1}^{n} w_j x_j.$$

Other definition of scalar product in two dimensional case is

$$\langle w, x \rangle = \|w\| \|x\| \cos(w, x)$$

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

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

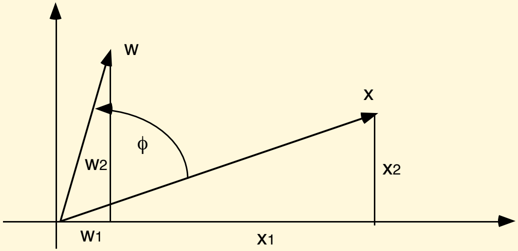

Figure 5: $w = (w_1, w_2)^T$ and $x = (x_1, x_2)^T$.

Lemma 1. *The following property holds*

$$\begin{aligned}\langle w, x\rangle &= w_1x_1 + w_2x_2\\ &= \sqrt{w_1^2 + w_2^2}\sqrt{x_1^2 + x_2^2}\cos(w, x)\\ &= \|w\|\|x\|\cos(w, x).\end{aligned}$$

Proof

$$\cos(w,x) = \cos((w, \text{1-st axis}) - (x, \text{1-st axis}))$$
$$= \cos((w, \text{1-st axis})\cos(x, \text{1-st axis}))+$$
$$\sin(w, \text{1-st axis})\sin(x, \text{1-st axis}) =$$
$$\frac{w_1x_1}{\sqrt{w_1^2+w_2^2}\sqrt{x_1^2+x_2^2}} + \frac{w_2x_2}{\sqrt{w_1^2+w_2^2}\sqrt{x_1^2+x_2^2}}$$

That is,

$$\|w\|\|x\|\cos(w,x) = \sqrt{w_1^2+w_2^2}\sqrt{x_1^2+x_2^2}\cos(w,x)$$
$$= w_1x_1 + w_2x_2.$$

From $\cos\pi/2 = 0$ it follows that $\langle w, x\rangle = 0$ whenever w and x are perpendicular.

If $\|w\| = 1$ (we say that w is normalized) then $|\langle w, x\rangle|$ is nothing else but the projection of x onto the direction of w. Really, if $\|w\| = 1$ then we get

$$\langle w, x\rangle = \|w\|\|x\| \cos(w, x) = \|x\| \cos(w, x)$$

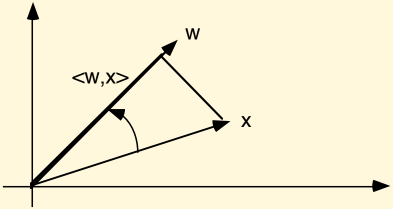

Figure 6: Projection of x onto the direction of w.

The problem of learning in neural networks is simply the problem of finding a set of connection strengths (weights) which allow the network to carry out the desired computation. The network is provided with a set of example input/output pairs (a training set) and is to modify its connections in order to approximate the function from which the input/output pairs have been drawn. The networks are then tested for ability to generalize.

2. The perception learning rule

The error correction learning procedure is simple enough in conception. The procedure is as follows: During training an input is put into the network and flows through the network generating a set of values on the output units. Then, the actual output is compared with the desired target, and a match is computed. If the output and target match, no change is made to the net. However, if the output differs from the target a change must be made to some of the connections.

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

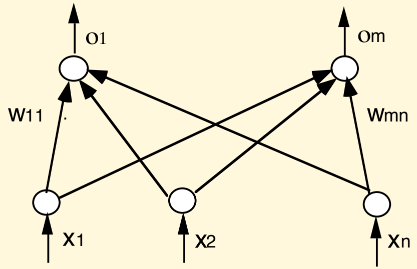

Figure 7: Single-layer feedforward network.

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.	$x^1 = (x_1^1, \dots x_n^1)$	$y^1 = (y_1^1, \dots, y_m^1)$
$\vdots$	$\vdots$	$\vdots$
K.	$x^K = (x_1^K, \dots x_n^K)$	$y^K = (y_1^K, \dots, y_m^K)$

Our problem is to find weight vectors w_i such that

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

for all training patterns k.

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

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

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

$$w_i := w_i + \eta(y_i - o_i)x,$$

for $i = 1, \dots, m$.

That is,

$$w_{ij} := w_{ij} + \eta(y_i - o_i)x_j,$$

for $j = 1, \dots, n$, where $\eta > 0$ is the learning rate.

From this equation it follows that if the desired output is equal to the computed output, $y_i = o_i$, 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, 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. We are given a training set of input/output pairs

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

Then the input components of the training patterns can be classified into two disjunct classes

$$C_1 = \{x^k | y^k = 1\},$$
$$C_2 = \{x^k | y^k = 0\}$$

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 $(x, 0)$.

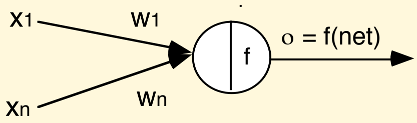

Figure 8: A simple processing element.

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

$$\langle w, x\rangle > 0 \text{ for each } x \in C_1$$

and

$$\langle w, x\rangle < 0 \text{ for each } x \in C_2.$$

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

If the problem is linearly separable then we can always suppose that the line separating the classes crosses the origin.

And the problem can be further transformed to the problem of finding a line, for which all the elements of $C_1 \cup (-C_2)$ can be found on the positive half-plane.

So, we search for a weight vector w such that $\langle w, x\rangle > 0$ for each $x \in C_1$, and $\langle w, x\rangle < 0$ for each $x \in C_2$. That is, $\langle w, x\langle >$

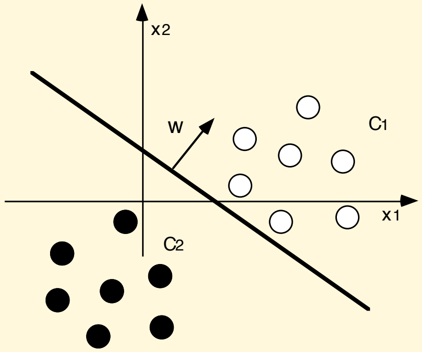

Figure 9: A two-class *linearly separable* classification problem.

0 for each $x \in C_1$ and $\langle w, -x\rangle > 0$ for each $-x \in C_2$. Which can be written in the form, $\langle w, x\rangle >$ for each $x \in C_1 \cup -C_2$.

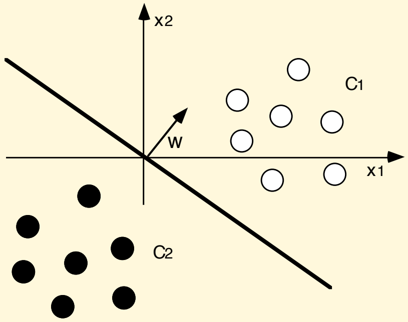

Figure 10: Shifting the origin.

And the weight adjustment in the perceptron learning rule is performed by

$$w_j := w_j + \eta(y - o)x_j.$$

where $\eta > 0$ is the learning rate, $y = 1$ is he desired output,

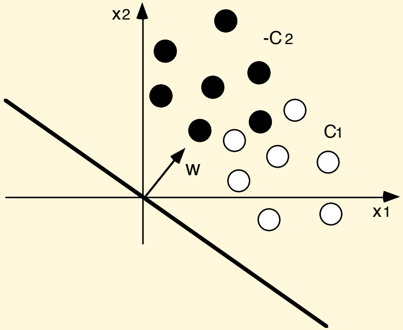

Figure 11: $C_1 \cup (-C_2)$ is on the postive side of the line.

$o \in \{0, 1\}$ is the computed output, x is the actual input to the neuron.

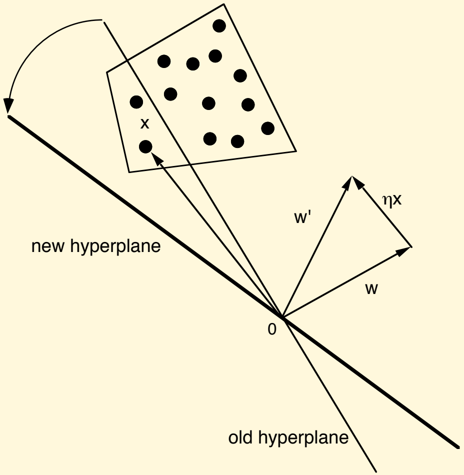

Figure 12: Illustration of the perceptron learning algorithm.

3. The perceptron learning algorithm

Given are K training pairs arranged in the training set

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

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

- **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 $o = o(x)$ is computed

$$o_i = \begin{cases} 1 & \text{if} < w_i, x > > 0 \\ 0 & \text{if} < w_i, x > < 0 \end{cases}$$

- **Step 4** Weights are updated

$$w_i := w_i + \eta(y_i - o_i)x,\ i = 1, \dots, m$$

- **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**

The following theorem shows that if the problem has solutions then the perceptron learning algorithm will find one of them.

Theorem 3.1. *(Convergence theorem) If the problem is linearly separable then the program will go to* **Step 3** *only finetely many times.*

4. 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

$$
\begin{aligned}
< w^6, x_1 >= -0.2 < 0, \\
< w^6, x_2 >= 1.70 > 0, \\
< w^6, x_3 >= 0.75 > 0.
\end{aligned}
$$

Minsky and Papert (1969) provided a very careful analysis of conditions under which the perceptron learning rule is capable of carrying out the required mappings.

They showed that the perceptron can not succesfully solve the problem

	x_1	x_1	$o(x_1, x_2)$
1.	1	1	0
2.	1	0	1
3.	0	1	1
4.	0	0	0

This Boolean function is know in the literature az exclusive (XOR).

Sometimes we call the above function as two-dimensional parity function.

The n-dimensional parity function is a binary Boolean function, which takes the value 1 if we have odd number of 1-s in the input vector, and zero otherwise.

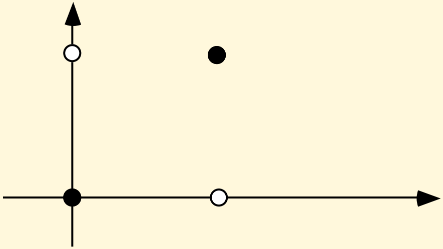

Figure 13: Illustration of exclusive or.

For example, the following function is the 3-dimensional parity function

	x_1	x_1	x_3	$o(x_1, x_2, x_3)$
1.	1	1	1	1
2.	1	1	0	0
3.	1	0	1	0
4.	1	0	0	1
5.	0	0	1	1
6.	0	1	1	0
7.	0	1	0	1
8.	0	0	0	0