Def 1. Un perceptron formal este un 4-uplu

$$(W, P, G_W, f_P)$$

unde $W \subset \mathbb{R}$, $P \subset \mathbb{R}$ sunt mulţimi finite de parametri iar:

– $G_W : \mathbb{R}^N \to \mathbb{R}$ este o funcţie ce depinde de parametrii W şi asociază unui vector construit din semnalele de intrare o valoare care reprezintă starea neuronului;

– $f_P : \mathbb{R} \to \mathbb{R}$ este o funcţie care depinde de parametrii P şi asociază stării curente a ~~neuronului~~ perceptronului o valoare care reprezintă semnalul de ieşire.

Observaţii 1. Funcţia G_W este numită funcţie de integrare a perceptronului, iar f_P este numită funcţie de transfer.

2. Parametrii W ponderează, de regulă, semnalele de intrare primite din partea perceptronilor cu care este conectat perceptronul curent, motiv pentru care sunt denumite ponderi ale conexiunilor.

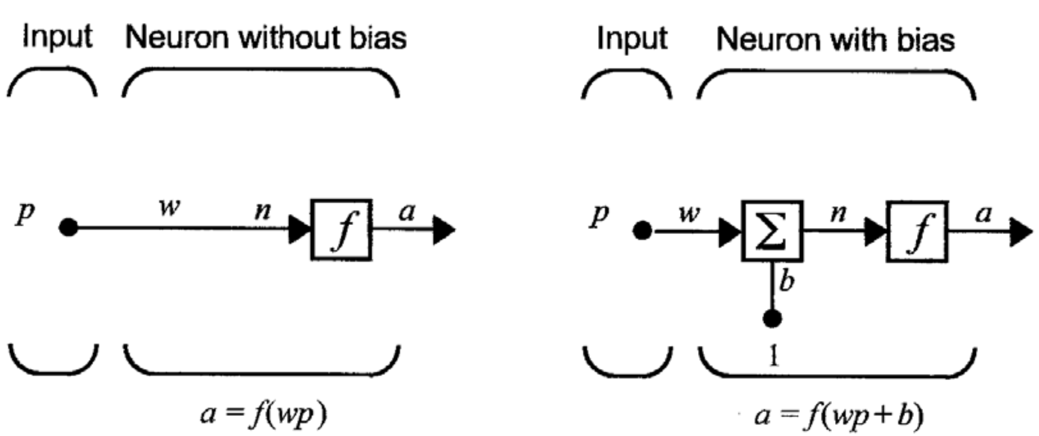
Exemple de funcţii de integrare. Presupunem că perceptronul analizat este conectat cu alţi N neuroni iar $\{z_j, j=\overline{1,N}\}$ reprezintă semnalele de intrare primite din partea acestor neuroni. Funcţiile de integrare cele mai des utilizate sunt de forma:

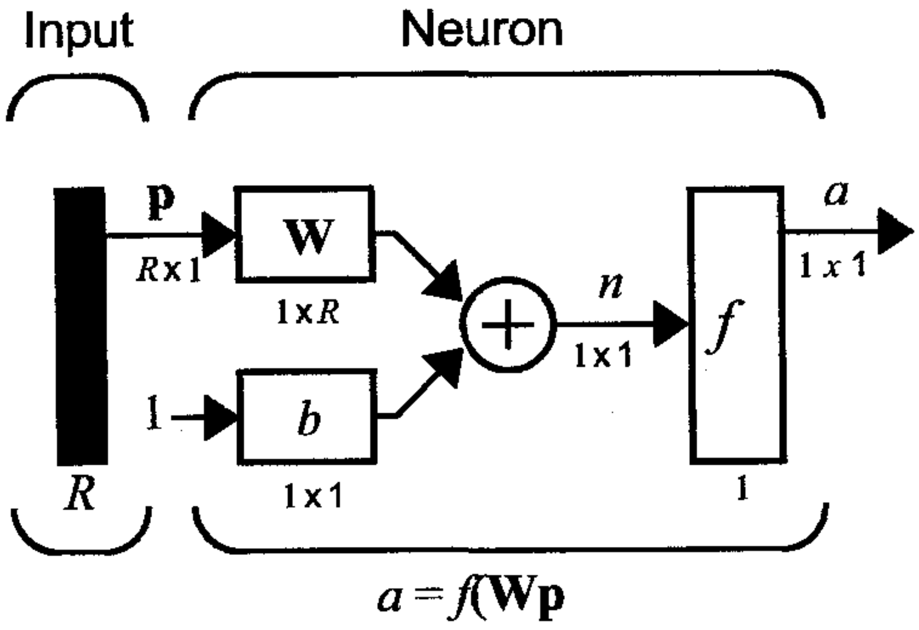
$$G_W(z_1, \ldots, z_N) = \sum_{j=1}^{N} w_j^{(1)} z_j + \sum_{j_1, j_2 = 1}^{N} w_{j_1 j_2}^{(2)} z_{j_1} z_{j_2} + \ldots + \sum_{j_1, j_2, \ldots, j_k = 1}^{N} w_{j_1 \ldots j_k}^{(k)} z_{j_1} \ldots z_{j_k}$$

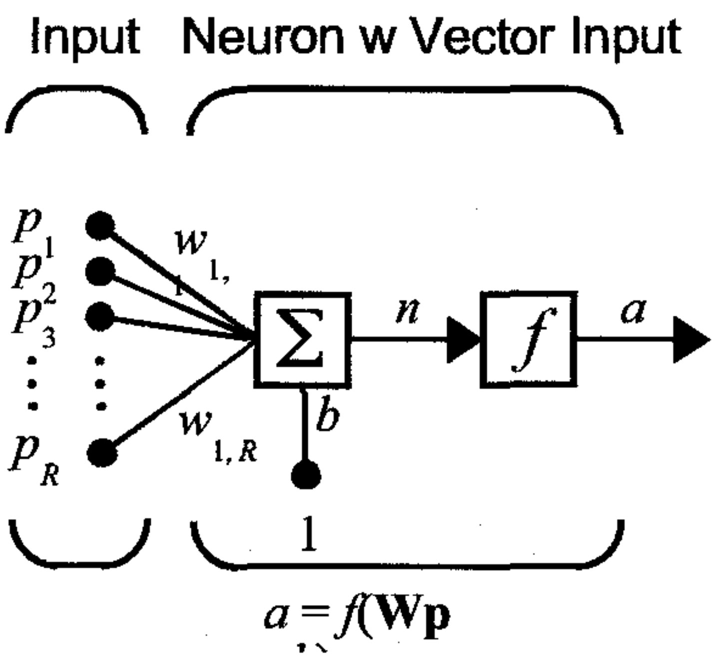
cu mulțimea parametrilor $W = \{ w_j^{(1)}, w_{j_1 j_2}^{(2)}, \ldots, w_{j_1 \ldots j_k}^{(k)} ; j, j_1 \ldots j_k = \overline{1, N} \}$.

În acest caz despre conexiunile dintre perceptroni se spune că sunt de ordin k.

Cazul particular cel mai reprezentativ este cel al conexiunilor de ordin 1 ($k=1$) pentru care G_W este o aplicație liniară.


Input
Neuron without bias
p
w
n
f
a
a = f(wp)
Input
Neuron with bias
p
w
Σ
n
f
a
b
1
a = f(wp+b)

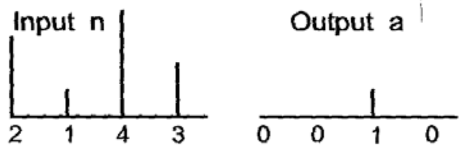

Input
Neuron
p
R x 1
W
1 x R
1
b
1 x 1
n
1 x 1
f
a
1 x 1
R
1
a = f(Wp



$$n = w_{1,1}p_1 + w_{1,2}p_2 + \ldots + w_{1,R}p_R + b$$

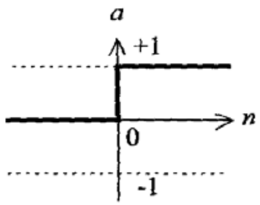
n = W*p + b

Transfer Function Graphs



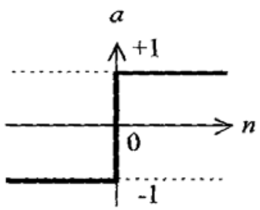
a = softmax(n)

Compet Transfer Function



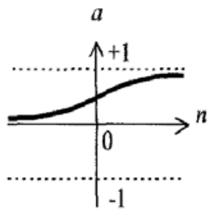
a = hardlim(n)

Hard-Limit Transfer Function



a = hardlims(n)

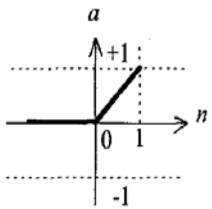
Symmetric Hard-Limit Trans. Funct.



a = logsig(n)

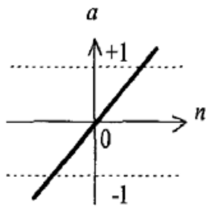
Log-Sigmoid Transfer Function

$$a = \frac{1}{1+\exp(-n)}$$



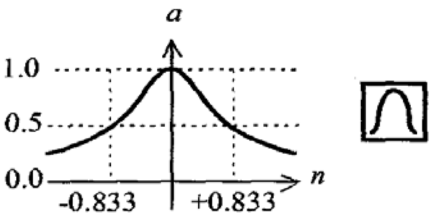
a = poslin(n)

Positive Linear Transfer Funct.



a = purelin(n)

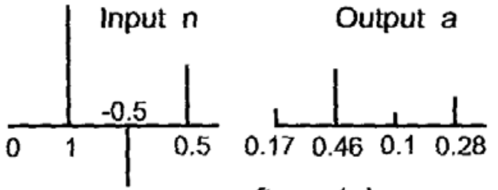
Linear Transfer Function



$a = radbas(n)$

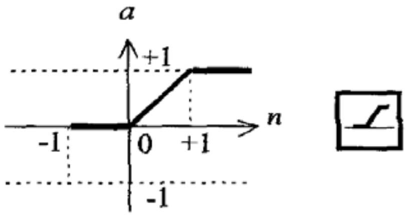
Radial Basis Function

$a = \exp(-n^2)$



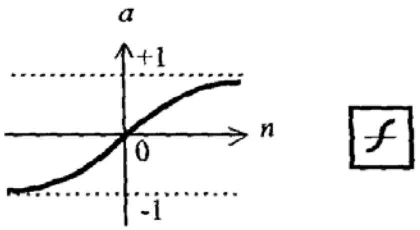
$a = softmax(n)$

Softmax Transfer Function S



$a = satlin(n)$

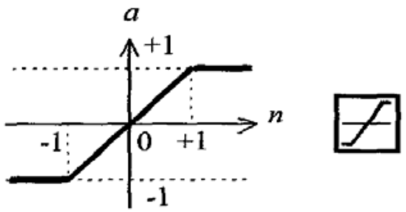
Satlin Transfer Function



$a = tansig(n)$

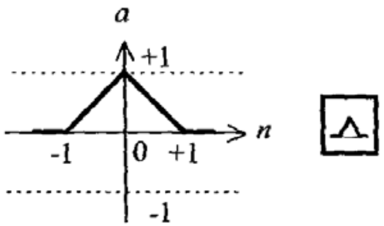
Tan-Sigmoid Transfer Function

$a = \dfrac{2}{1+\exp(-2n)}$



$a = satlins(n)$

Satlins Transfer Function



$a = tribas(n)$

Triangular Basis Function

Simple neuron and transfer functions

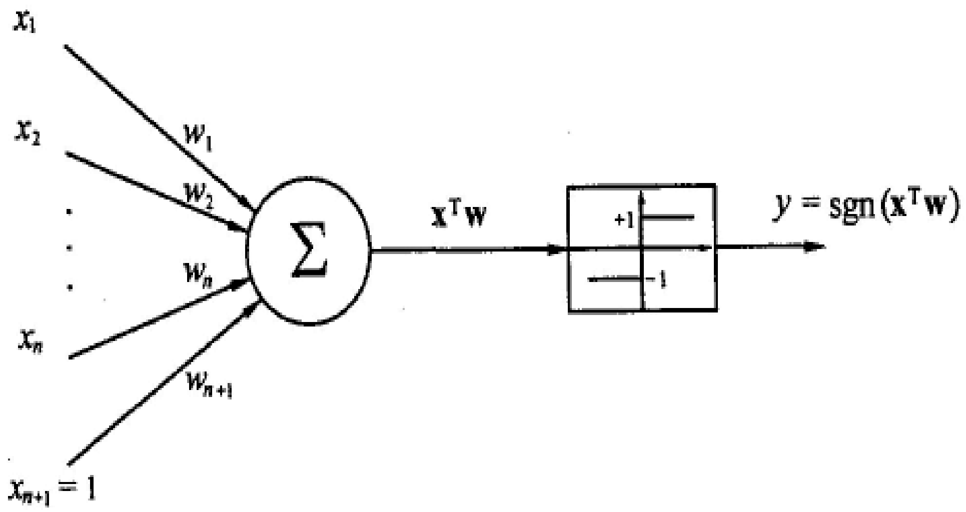
nnd2n1 One-input neuron demonstration.

Neuron with vector input

nnd2n2 Two-input neuron demonstration.

Perceptron Learning Rule

Consider the linear threshold gate –LTG- shown in Figure 1, which will be referred to as the *perceptron.* The perceptron maps an input vector $\mathbf{x} = [x_1 \ x_2 \ ... \ x_{n+1}]^{\mathrm{T}}$ to a bipolar binary output y, and thus it may be view, as a simple two-class classifier. The input signal x_{n+1} is usually set to 1 and plays the role of a bias to the perceptron. We will denote by $\mathbf{w}$ the vector $\mathbf{w} = [w_1 \ w_2 \ ... \ w_{n+1}]^{\mathrm{T}} \in R^{n+1}$ consisting of the free parameters (weights) of the perceptron. The input/output relation for the perceptron is given by $y = \mathrm{sgn}(\mathbf{x}^T\mathbf{w})$, where sgn is the "sign" function, which returns +1 or –1 depending on whether the sign of its scalar argument is positive or negative, respectively.



Assume we are training this perceptron to load (learn) the training pairs $\{\mathbf{x}^1, d^1\}, \{\mathbf{x}^2, d^2\}, \ldots, \{\mathbf{x}^m, d^m\}$, where $\mathbf{x}^k \in R^{n+1}$ is the kth input vector and $d^k \in \{-1, +1\}$, k =1, 2, ... , m, is the desired target for the kth input vector (usually the order of these training pairs is random). The entire collection of these pairs is called the *training set.*

The goal, then, is to design a perceptron such that for each input vector $\mathbf{x}^k$ of the training set, the perceptron output y^k matches the desired target d^k; that is, we require $y^k = \mathrm{sgn}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^k\right) = d^k$, for each k =1, 2, ... , m. In this case we say that the perceptron correctly classifies the training set. Of course, "designing" an appropriate perceptron to correctly classify the training set amounts to determining a weight vector $\mathbf{w}^*$ such that the following relations are satisfied:

$$\begin{cases} \left(\mathbf{x}^k\right)^{\mathrm{T}} \mathbf{w}^* > 0 & \text{if } d^k = +1 \\ \left(\mathbf{x}^k\right)^{\mathrm{T}} \mathbf{w}^* < 0 & \text{if } d^k = -1 \end{cases} \qquad (2.1)$$

Recall that the set of all $\mathbf{x}$ which satisfy $\mathbf{x}^{\mathrm{T}}\,\mathbf{w}^{*}=0$ defines a hyperplane in R^{n}. Thus, in the context of the preceding discussion, finding a solution vector $\mathbf{w}^{*}$ to Equation (2.1) is equivalent to finding a separating hyperplane that correctly classifies all vectors $\mathbf{x}^{k}$, k = 1, 2, ... , m. In other words, we desire a hyperplane $\mathbf{x}^{\mathrm{T}}\,\mathbf{w}^{*}=0$ that partitions the input space into two distinct regions, one containing all points $\mathbf{x}^{k}$ with $d^{k}=+1$ and the other region containing all points $\mathbf{x}^{k}$ with $d^{k}=-1$.

One possible incremental method for arriving at a solution $\mathbf{w}^{*}$ is to invoke the perceptron learning rule (Rosenblatt, 1962):

$$\begin{cases} \mathbf{w}^{1} \ \text{arbitrary} \\ \mathbf{w}^{k+1}=\mathbf{w}^{k}+\rho\left(d^{k}-y^{k}\right)\mathbf{x}^{k} \quad k=1,2,\ldots \end{cases} \tag{2.2}$$

where ρ is a positive constant called the *learning rate.* The incremental learning process given in Equation (2.2) proceeds as follows: First, an initial weight vector $\mathbf{w}^{1}$ is selected (usually at random) to begin the process. Then, the m pairs $\{\mathbf{x}^k, d^k\}$ of the training set are used to successively update the weight vector until (hopefully) a solution $\mathbf{w}^*$ is found that correctly classifies the training set. This process of sequentially presenting the training patterns is usually referred to as *cycling* through the training set, and a complete presentation of the m training pairs is referred to as a *cycle* (or *pass*) through the training set. In general, more than one cycle through the training set is required to determine an appropriate solution vector. Hence, in Equation (2.2), the superscript k in $\mathbf{w}^k$ refers to the iteration number. On the other hand, the superscript k in $\mathbf{x}^k$ (and d^k) is the label of the training pair presented at the kth iteration. To be more precise, if the number of training pairs m is finite, then the superscripts in $\mathbf{x}^k$ and d^k should be replaced by $\left[(k-1) \bmod m\right]+1$. Here, a mod b returns the remainder of the division of a by b (e.g., 5 mod 8 = 5, 8 mod 8 = 0, and 19 mod 8 = 3). This observation is valid for all incremental learning rules presented in this chapter.

Decision Boundaries

nnd4db Decision boundaries demonstration.

Notice that for $\rho = 0.5$, the perceptron learning rule can be written as

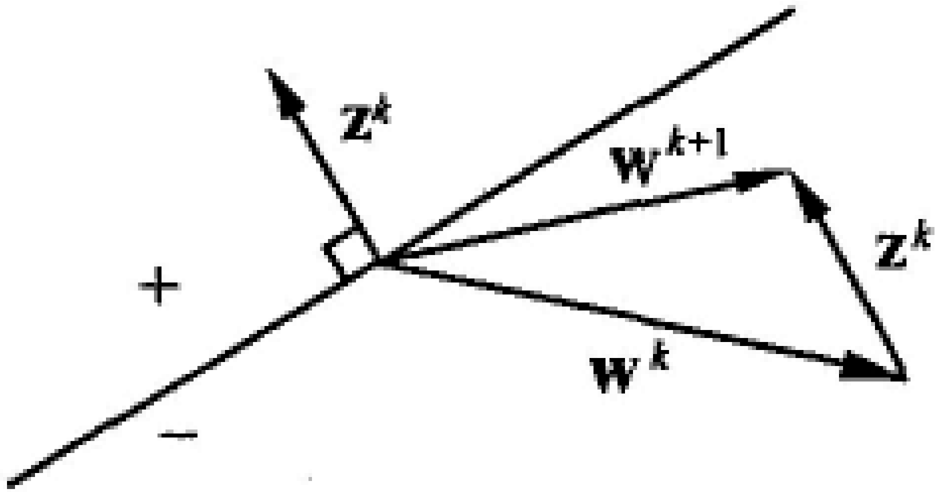
$$\begin{cases} \mathbf{w}^1 \quad \text{arbitrary} \\ \mathbf{w}^{k+1} = \mathbf{w}^k + \mathbf{z}^k \quad \text{if } \left(\mathbf{z}^k\right)^{\mathrm{T}} \mathbf{w}^k \le 0 \\ \mathbf{w}^{k+1} = \mathbf{w}^k \quad \text{otherwise} \end{cases} \tag{2.3}$$

where $$\mathbf{z}^k = \begin{cases} +\mathbf{x}^k & \text{if } d^k = +1 \\ -\mathbf{x}^k & \text{if } d^k = -1 \end{cases} \tag{2.4}$$

That is, a correction is made if and only if a misclassification, indicated by

$$\left(\mathbf{z}^k\right)^{\mathrm{T}} \mathbf{w}^k \le 0 \tag{2.5}$$

occurs. The addition of vector $\mathbf{z}^k$ to $\mathbf{w}^k$ in Equation (2.3) moves the weight vector directly toward and perhaps across the hyperplane $\left(\mathbf{z}^k\right)^{\mathrm{T}} \mathbf{w}^k = 0$. The new inner product $\left(\mathbf{z}^k\right)^{\mathrm{T}} \mathbf{w}^{k+1}$ is larger than $\left(\mathbf{z}^k\right)^{\mathrm{T}} \mathbf{w}^k$ by the amount of $\left\|\mathbf{z}^k\right\|^2$, and the correction $\Delta \mathbf{w}^k = \mathbf{w}^{k+1} - \mathbf{w}^k$ is clearly moving $\mathbf{w}^k$ in a good direction, the direction of increasing $\left(\mathbf{z}^k\right)^{\mathrm{T}} \mathbf{w}^k$, as can be seen from Figure 2-2.[1]. Thus the perceptron learning rule attempts to find a solution $\mathbf{w}^*$ for the following system of inequalities:



$$\left(\mathbf{z}^k\right)^{\mathrm{T}} \mathbf{w} > 0 \qquad \text{for } k = 1, 2, \ldots, m \tag{2.6}$$

[1] The quantity $\|\mathbf{z}\|^2$ is given by $\mathbf{z}^{\mathrm{T}}\mathbf{z}$ and is sometimes referred to as the *energy* of **z**. $\|\mathbf{z}\|$ is the Euclidean norm (length) of vector **z** and is given by the square root of the sum of the squares of the components of **z** [note that $\|\mathbf{z}\| = \|\mathbf{x}\|$ by virtue of Equation (2.4)].

In an analysis of any learning algorithm, and in particular the perceptron learning algorithm of Equation (2.2), there are two main issues to consider:

(1) the existence of solutions and

(2) convergence of the algorithm to the desired solutions (if they exist).

In the case of the perceptron, it is clear that a solution vector (i.e., a vector $\mathbf{w}^*$ that correctly classifies the training set) exists if and only if the given training set is linearly separable. Assuming, then, that the training set is linearly separable, we may proceed to show that the perceptron learning rule converges to a solution (Novikoff, 1962; Nilsson, 1965) as follows: Let $\mathbf{w}^*$ be any solution vector so that

$$\left(\mathbf{z}^k\right)^{\mathrm{T}} \mathbf{w}^k > 0 \qquad \text{for } k = 1, 2, \ldots, m \qquad (2.7)$$

Then, from Equation (2.3), if the *k*th pattern is misclassified, we may write

$$\mathbf{w}^{k+1} - \alpha\mathbf{w}^* = \mathbf{w}^k - \alpha\mathbf{w}^* + \mathbf{z}^k \qquad (2.8)$$

where α is a positive scale factor, and hence

$$\left\|\mathbf{w}^{k+1}-\alpha\mathbf{w}^*\right\|^2=\left\|\mathbf{w}^k-\alpha\mathbf{w}^*\right\|^2+2\left(\mathbf{z}^k\right)^{\mathrm{T}}\left(\mathbf{w}^k-\alpha\mathbf{w}^*\right)+\left\|\mathbf{z}^k\right\|^2 \tag{2.9}$$

Since $\mathbf{z}^k$ is misclassified, we have $\left(\mathbf{z}^k\right)^{\mathrm{T}}\mathbf{w}^k\leq 0$, and thus

$$\left\|\mathbf{w}^{k+1}-\alpha\mathbf{w}^*\right\|^2\leq\left\|\mathbf{w}^k-\alpha\mathbf{w}^*\right\|^2-2\alpha\left(\mathbf{z}^k\right)^{\mathrm{T}}\mathbf{w}^*+\left\|\mathbf{z}^k\right\|^2 \tag{2.10}$$

Now, let $\beta^2=\max_i\left\|\mathbf{z}^i\right\|^2$ and $\gamma=\min_i\left(\mathbf{z}^i\right)^{\mathrm{T}}\mathbf{w}^*$ [γ is positive because $\left(\mathbf{z}^i\right)^{\mathrm{T}}\mathbf{w}^*>0$] and substitute into Equation (2.10) to get

$$\left\|\mathbf{w}^{k+1}-\alpha\mathbf{w}^*\right\|^2\leq\left\|\mathbf{w}^k-\alpha\mathbf{w}^*\right\|^2-2\alpha\gamma+\beta^2 \tag{2.11}$$

If we choose α sufficiently large, in particular $\alpha=\beta^2/\gamma$, we obtain

$$\left\|\mathbf{w}^{k+1}-\alpha\mathbf{w}^*\right\|^2\leq\left\|\mathbf{w}^k-\alpha\mathbf{w}^*\right\|^2-\beta^2 \tag{2.12}$$

Thus the square distance between $\mathbf{w}^k$ and $\alpha\mathbf{w}^k$ is reduced by at least β^2 at each correction, and after k corrections, we may write Equation (2.12) as

$$0 \le \left\| \mathbf{w}^{k+1} - \alpha\mathbf{w}^* \right\|^2 \le \left\| \mathbf{w}^1 - \alpha\mathbf{w}^* \right\|^2 - k\beta^2 \tag{2.13}$$

It follows that the sequence of corrections must terminate after no more than k_0 corrections, where

$$k_0 = \frac{\left\| \mathbf{w}^1 - \alpha\mathbf{w}^* \right\|^2}{\beta^2} \tag{2.14}$$

Therefore, if a solution exists, it is achieved in a finite number of iterations. When corrections cease, the resulting weight vector must classify all the samples correctly, since a correction occurs whenever a sample is misclassified, and since each sample appears infinitely often in the sequence. In general, a linearly separable problem admits an infinite number of solutions. The perceptron learning rule in Equation (2.2) converges to one of these solutions. This solution, though, is sensitive to the value of the learning rate ρ used and to the order of presentation of the training pairs.

This sensitivity is responsible for the varying quality of the perception-generated separating surface observed in simulations.

The bound on the number of corrections k_0 given by Equation (2.14) depends on the choice of the initial weight vector $\mathbf{w}^1$. If $\mathbf{w}^1 = 0$, we get

$$k_0 = \frac{\alpha^2 \left\|\mathbf{w}^*\right\|^2}{\beta^2} = \frac{\beta^2 \left\|\mathbf{w}^*\right\|^2}{\gamma^2} \quad \text{or} \quad k_0 = \frac{\max_i \left\|\mathbf{x}^i\right\|^2 \left\|\mathbf{w}^*\right\|^2}{\left[\min_i \left(\mathbf{x}^i\right)^{\mathrm{T}} \mathbf{w}^*\right]^2} \qquad (2.15)$$

Here, k_0 is a function of the initially unknown solution weight vector $\mathbf{w}^*$. Therefore, Equation (2.15) is of no help for predicting the maximum number of corrections. However, the denominator of Equation (2.15) implies that the difficulty of the problem is essentially determined by the samples most nearly orthogonal to the solution vector.

Perceptron learning rule

nnd4pr Perceptron rule demonstration