

## NN II: The delta 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.**

Let's first recall the definition of derivative of single-variable functions.

**Definition 1.** *The derivative of  $f$  at (an interior point of its domain)  $x$ , denoted by  $f'(x)$ , and defined by*

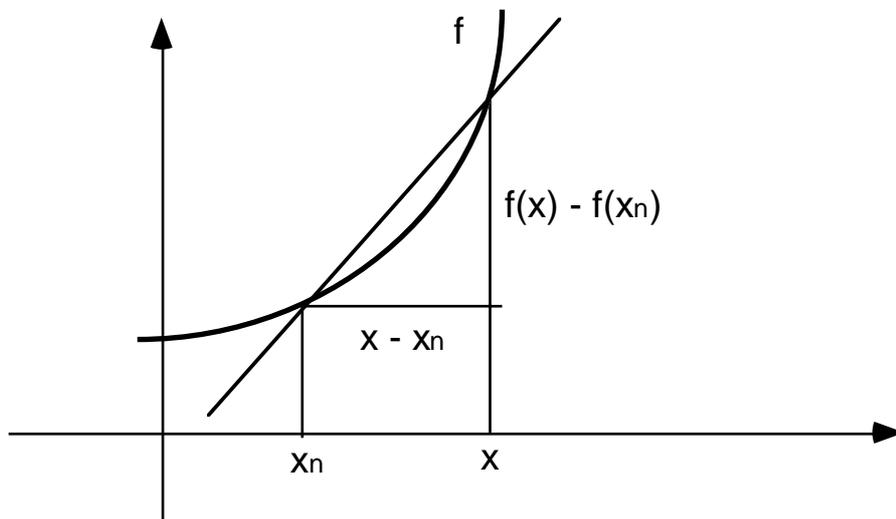
$$f'(x) = \lim_{x_n \rightarrow x} \frac{f(x) - f(x_n)}{x - x_n}$$

Let us consider a differentiable function

$$f: \mathbb{R} \rightarrow \mathbb{R}.$$

The derivative of  $f$  at (an interior point of its domain)  $x$  is denoted by  $f'(x)$ .

If  $f'(x) > 0$  then we say that  $f$  is increasing at  $x$ , if  $f'(x) < 0$  then we say that  $f$  is decreasing at  $x$ , if  $f'(x) = 0$  then  $f$  can have a local maximum, minimum or inflexion point at  $x$ .



Derivative of function  $f$ .

A differentiable function is always increasing in the direction of its derivative, and decreasing in the opposite direction.

It means that if we want to find one of the local minima of a function  $f$  starting from a point  $x^0$  then we should search for a second candidate:

- in the right-hand side of  $x^0$  if  $f'(x^0) < 0$ , when

$f$  is decreasing at  $x^0$ ;

- in the left-hand side of  $x^0$  if  $f'(x^0) > 0$ , when  $f$  increasing at  $x^0$ .

The equation for the line crossing the point  $(x^0, f(x^0))$  is given by

$$\frac{y - f(x^0)}{x - x^0} = f'(x^0)$$

that is

$$y = f(x^0) + (x - x^0)f'(x^0)$$

The next approximation, denoted by  $x^1$ , is a solution to the equation

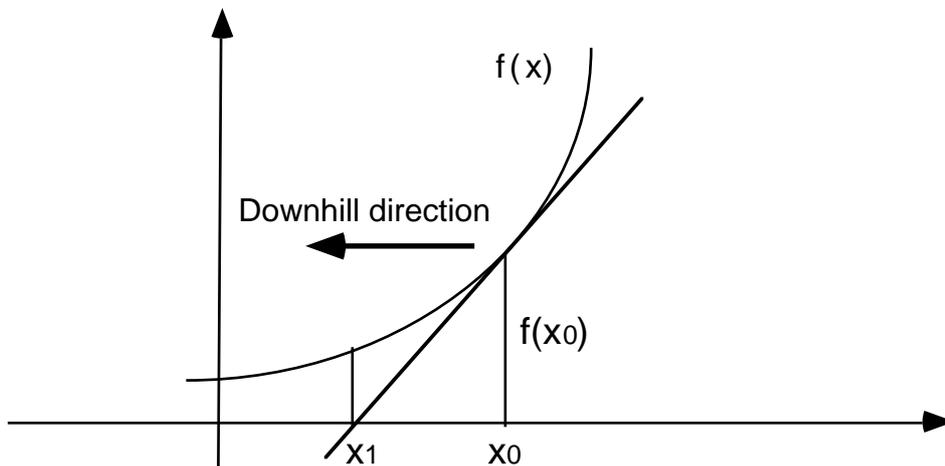
$$f(x^0) + (x - x^0)f'(x^0) = 0$$

which is,

$$x^1 = x^0 - \frac{f(x^0)}{f'(x^0)}$$

This idea can be applied successively, that is

$$x^{n+1} = x^n - \frac{f(x^n)}{f'(x^n)}.$$



The downhill direction is negative at  $x_0$ .

The above procedure is a typical descent method. In a descent method the next iteration  $w^{n+1}$  should satisfy the following property

$$f(w^{n+1}) < f(w^n)$$

i.e. the value of  $f$  at  $w^{n+1}$  is smaller than its previous value at  $w^n$ .

In error correction learning procedure, each iteration of a descent method calculates the downhill direction (opposite of the direction of the deriva-

tive) at  $w^n$  which means that for a sufficiently small  $\eta > 0$  the inequality

$$f(w^n - \eta f'(w^n)) < f(w^n)$$

should hold, and we let  $w^{n+1}$  be the vector

$$w^{n+1} = w^n - \eta f'(w^n)$$

Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be a real-valued function. In a descent method, whatever is the next iteration,  $w^{n+1}$ , it should satisfy the property

$$f(w^{n+1}) < f(w^n)$$

i.e. the value of  $f$  at  $w^{n+1}$  is smaller than its value at previous approximation  $w^n$ .

Each iteration of a descent method calculates a downhill direction (opposite of the direction of the derivative) at  $w^n$  which means that for a sufficiently small

$\eta > 0$  the inequality

$$f(w^n - \eta f'(w^n)) < f(w^n)$$

should hold, and we let  $w^{n+1}$  be the vector

$$w^{n+1} = w^n - \eta f'(w^n).$$

Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be a real-valued function and let  $e \in \mathbb{R}^n$  with  $\|e\| = 1$  be a given direction. The derivative of  $f$  with respect  $e$  at  $w$  is defined as

$$\partial_e f(w) = \lim_{t \rightarrow +0} \frac{f(w + te) - f(w)}{t}.$$

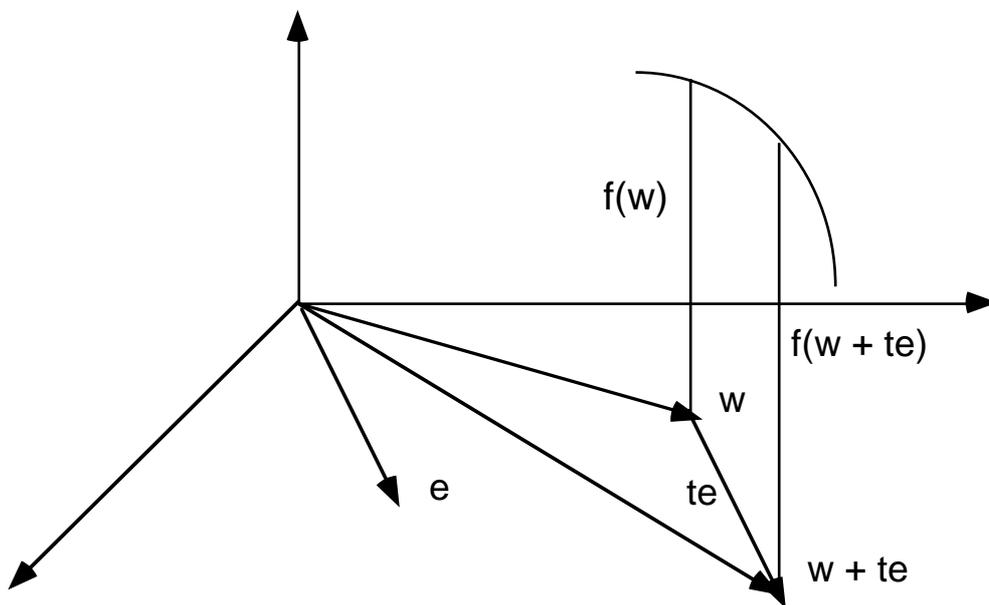
If

$$e = (0, \dots, \overbrace{1}^{i\text{-th}}, \dots, 0)^T,$$

i.e.  $e$  is the  $i$ -th basic direction then instead of  $\partial_e f(w)$

we write  $\partial_i f(w)$ , which is defined by

$$\partial_i f(w) = \lim_{t \rightarrow +0} \frac{f(w_1, \dots, w_i + t, \dots, w_n) - f(w_1, \dots, w_i, \dots, w_n)}{t}.$$



The derivative of  $f$  with respect to the direction  $e$ ..

The gradient of  $f$  at  $w$ , denoted by  $f'(w)$  is defined by

$$f'(w) = (\partial_1 f(w), \dots, \partial_n f(w))^T$$

**Example 1.** Let  $f(w_1, w_2) = w_1^2 + w_2^2$  then the gradient of  $f$  is given by

$$f'(w) = 2w = (2w_1, 2w_2)^T.$$

The gradient vector always points to the uphill direction of  $f$ . The downhill (steepest descent) direction of  $f$  at  $w$  is the opposite of the uphill direction, i.e. the downhill direction is  $-f'(w)$ , which is

$$(-\partial_1 f(w), \dots, -\partial_n f(w))^T.$$

**Definition 2.** A linear activation function is a mapping from  $f: \mathbb{R} \rightarrow \mathbb{R}$  such that

$$f(t) = t$$

for all  $t \in \mathbb{R}$ .

Suppose we are given a single-layer network with  $n$  input units and  $m$  linear output units, i.e. the output of the  $i$ -th neuron can be written as

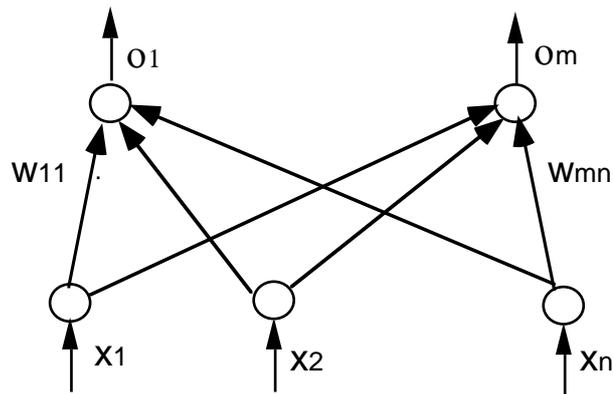
$$o_i = net_i = \langle w_i, x \rangle = w_{i1}x_1 + \dots + w_{in}x_n,$$

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

Assume we have the following training set

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

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



Single-layer feedforward network with  $m$  output units

The basic idea of the delta learning rule is to define a measure of the overall performance of the system and then to find a way to optimize that performance.

In our network, we can define the performance of the system as

$$E = \sum_{k=1}^K E_k = \frac{1}{2} \sum_{k=1}^K \|y^k - o^k\|^2$$

That is

$$\begin{aligned} E &= \frac{1}{2} \sum_{k=1}^K \sum_{i=1}^m (y_i^k - o_i^k)^2 \\ &= \frac{1}{2} \sum_{k=1}^K \sum_{i=1}^m (y_i^k - \langle w_i, x^k \rangle)^2 \end{aligned}$$

where  $i$  indexes the output units;  $k$  indexes the input/output pairs to be learned;  $y_i^k$  indicates the target for a particular output unit on a particular pattern;

$$o_i^k := \langle w_i, x^k \rangle$$

indicates the actual output for that unit on that pattern; and  $E$  is the total error of the system.

The goal, then, is to minimize this function.

It turns out, if the output functions are differentiable, that this problem has a simple solution: namely, we can assign a particular unit blame in proportion to the degree to which changes in that unit's activity lead to changes in the error.

That is, we change the weights of the system in proportion to the derivative of the error with respect to the weights.

The rule for changing weights following presentation of input/output pair

$$(x, y)$$

(we omit the index  $k$  for the sake of simplicity) is given by the gradient descent method, i.e. we min-

imize the quadratic error function by using the following iteration process

$$w_{ij} := w_{ij} - \eta \frac{\partial E_k}{\partial w_{ij}}$$

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

Let us compute now the partial derivative of the error function  $E_k$  with respect to  $w_{ij}$

$$\frac{\partial E_k}{\partial w_{ij}} = \frac{\partial E_k}{\partial net_i} \frac{\partial net_i}{\partial w_{ij}} = -(y_i - o_i)x_j$$

where  $net_i = w_{i1}x_1 + \dots + w_{in}x_n$ .

That is,

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

for  $j = 1, \dots, n$ .

**Definition 3.** *The error signal term, denoted by  $\delta_i^k$  and called delta, produced by the  $i$ -th output neuron is defined as*

$$\delta_i = -\frac{\partial E_k}{\partial net_i} = (y_i - o_i)$$

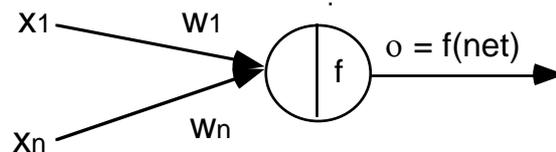
*For linear output units  $\delta_i$  is nothing else but the difference between the desired and computed output values of the  $i$ -th neuron.*

So the delta learning rule can be written as

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

where  $w_i$  is the weight vector of the  $i$ -th output unit,  $x$  is the actual input to the system,  $y_i$  is the  $i$ -th coordinate of the desired output,  $o_i$  is the  $i$ -th coordinate of the computed output and  $\eta > 0$  is the learning rate.

If we have only one output unit then the delta learning rule collapses into



A single neuron net.

$$w := w + \eta(y - o)x = w + \eta\delta x$$

where  $\delta$  denotes the difference between the desired and the computed output.

A key advantage of neural network systems is that these simple, yet powerful learning procedures can be defined, allowing the systems to adapt to their environments.

*The essential character of such networks is that they map similar input patterns to similar output patterns.*

This characteristic is what allows these networks to make reasonable generalizations and perform reasonably on patterns that have never before been presented. The similarity of patterns in a connectionist system is determined by their overlap.

The overlap in such networks is determined outside the learning system itself whatever produces the patterns.

The standard delta rule essentially implements gradient descent in sum-squared error for linear activation functions.

It should be noted that the delta learning rule was introduced only recently for neural network training by McClelland and Rumelhart.

This rule parallels the discrete perceptron training rule. It also can be called the **continuous percep-**

## tron training rule.

**Summary 1.** *The delta learning rule with linear activation functions. Given are  $K$  training pairs arranged in the training set*

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

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

- **Step 1**  $\eta > 0$ ,  $E_{\max} > 0$  are chosen
- **Step 2** Weights  $w_{ij}$  are initialized at small random values,  $k := 1$ , and the running error  $E$  is set to 0
- **Step 3** Training starts here. Input  $x^k$  is presented,  $x := x^k$ ,  $y := y^k$ , and output  $o = (o_1, \dots, o_m)^T$  is computed

$$o_i = \langle w_i, x \rangle = w_i^T x$$

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

- **Step 4** Weights are updated

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

- **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

$$E < E_{\max}$$

terminate the training session. If

$$E > E_{\max}$$

then  $E$  is set to 0 and we initiate a new training cycle by going back to **Step 3**