Learning in Neural Networks

Under the notion of learning in a network, it is a procedure for modifying the weights and biases of network, also referred to as training algorithm, to force a network to yield a particular response to a specific input. Many learning rules are in use. Most of these rules are some sort of variation of the well-known and oldest learning rule, Hebb's Rule. Research into different learning functions continues as new ideas routinely show up in trade publications. Some researchers have the modelling of biological learning as their main objective. Others are experimenting with adaptations of their perceptions of how nature handles learning. Learning is certainly more complex than the simplifications represented by the learning rules used. Two different types of learning rules can be distinguished:

- Learning with supervision, and
- Learning without supervision

Supervised learning

Learning rule is provided with a set of input-output data (also called training data) of proper network behaviour. As the inputs are applied to the network, the network outputs are compared to the target outputs. The learning rule is then used to adjust the weights and biases of the network in order to move the network outputs closer to the targets. Supervised learning is illustrated in Figure 1.

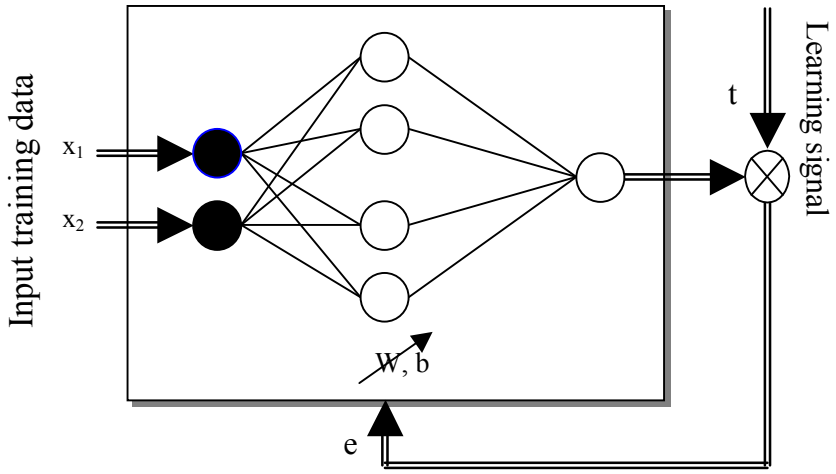


Figure 1: Supervised learning.

The learning method tries to minimize the current errors of all processing elements. This global error reduction is created over time by continuously modifying the weights until an acceptable error goal is reached. Training consists of presenting input and output data to the network. This data is often referred to as the training set. That is, for each input set provided to the system, the corresponding desired output set is provided as well.

$$\{^{1}(x_1, x_2), {}^{1}(t_1)\}, \{^{2}(x_1, x_2), {}^{2}(t_1)\}, \{^{3}(x_1, x_2), {}^{3}(t_1)\}, \ldots, \{^{N}(x_1, x_2), {}^{N}(t_1)\} \quad (1)$$

This training can consume a lot of time. In prototype systems, with inadequate processing power, learning can take days and even weeks. The rules that belong to supervised learning are

- Widrow-Hoff rule
- Gradient descent
  - Delata rule
  - Backpropagation rule

(i) Widrow-Hoff Rule

Widrow-Hoff learning rule (Widrow, 1962) is applicable for supervised training of neural networks. It is independent of the activation function of the neurons used since it minimises the squared error between the desired output and neuron's activation value. This rule can be considered a special case of delta learning rule.

(ii) Gradient Descent Rule

This rule is similar to the Delta rule in that the derivative of the transfer function is still used to modify the delta error before it is applied to the connection weights. Here, however, an additional proportional constant tied to the learning rate is added to the final modifying factor acting upon the weight. This rule is commonly used, even though the convergence to a stable point is very slowly. It has been shown that different learning rates for different layers of a network help the learning process converge faster.

(iii) Delta Rule

This rule is a variation of Hebb's rule and based on the simple idea of continuously modifying the weights of the input connections to reduce the difference (the delta) between the desired output and the actual output of the network. It changes the weights in such a way that minimizes the mean squared error of the network. The rule is also referred to as Widrow-Hoff learning rule and the Least Mean Square (LMS) learning rule. The delta error in the output layer is transformed by the derivative of the transfer function and this error is back-propagated into previous layers one layer at a time. The process of back-propagating the network errors continues until the first layer is reached. When using the delta rule, it is important to ensure that the input data set is well randomised. Ordered or structured presentation of the training data set can lead to a network which can not converge to the desired accuracy meaning that the network is incapable of learning the problem.

Unsupervised Learning

In unsupervised learning, the desired response is not known, i.e., no external learning signals to adjust network weights. Instead, they internally monitor their performance. Since no information is available as to correctness or incorrectness of response, learning looks for regularities or trends in the input signals, and makes adaptations according to the function of the network. Even without being told whether it's right or wrong, the network still must have some information about how to organize itself. This information is built into the network topology and learning rules.

An unsupervised learning algorithm might emphasize cooperation among clusters of processing elements. In such a scheme, the clusters would work together. Competition between processing elements could also form a basis for learning. Training of competitive clusters could amplify the responses of specific groups to specific stimuli. As such, it would associate those groups with each other and with a specific appropriate response. Normally, when competition for learning is in effect, only the weights belonging to the winning processing element will be updated.

Currently, this learning method is limited to networks known as self-organizing maps. At the present state of the art, unsupervised learning is not well understood and is still the subject of research. An unsupervised learning scheme is illustrated in Figure 2.

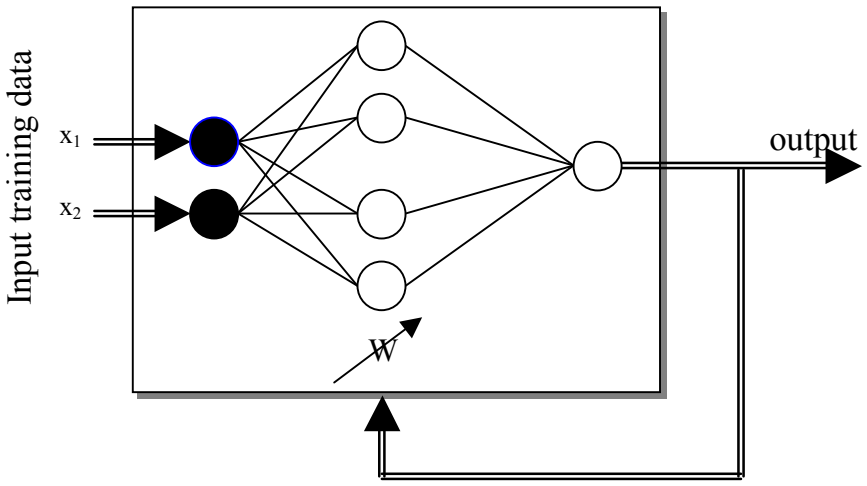


Figure 2: Unsupervised learning

The rules that belong to unsupervised learning are

- Hebb's rule
- Hopfield rule
- Kohonen's rule

(i) Hebb's Rule

The first, and undoubtedly the best known, learning rule was introduced by Donald Hebb in his book 'The Organization of Behavior' in 1949. This basic rule is: "If a neuron receives an input from another neuron, and if both are highly active (mathematically have the same sign), the weight between the neurons should be strengthened".

(ii) Hopfield rule

It is similar to Hebbian learning rule with the exception that it specifies the magnitude of the strengthening or weakening. It states, "if the desired output and the input are both active or both inactive, increment the connection weight by the learning rate, otherwise decrement the weight by the learning rate."

(v). Kohonen's Learning rule

This procedure, developed by Teuvo Kohonen, was inspired by learning in biological systems. In this procedure, the processing elements compete for the opportunity to learn, or update their weights. The processing element with the largest output is declared the winner and has the capability of inhibiting its competitors as well as exciting its neighbours. Only the winner is permitted an output, and only the winner plus its neighbours are allowed to adjust their connection weights. Further, the size of the neighbourhood can vary during the training period. The usual paradigm is to start with a larger definition of the neighbourhood, and narrowing as the training process proceeds. Because the winning element is defined as the one that has the closest match to the input pattern, Kohonen networks model the distribution of the inputs. This is good for statistical or topological modelling of the data and is sometimes referred to as self-organizing maps.

Detail description of these learning rules will be covered in later lectures. For better understanding of the materials, see (Zurada, 1992).

The above-mentioned rules will be covered in this module. It does not mean that these are the only learning rules for neural networks.

Widrow-Hoff learning algorithm

Widrow-Hoff learning rule (Widrow, 1962) is applicable for supervised training of neural networks. It minimises the squared error between the desired output d_i and neuron's actual output. The Widrow-Hoff learning rule is shown diagrammatically in Figure 3.

The weight vector increment under this learning rule is

$$\Delta w_i = -\eta(d_i - o_i)x \tag{2}$$

where η is the learning constant. The output o_i is defined as

$$o_i = f(net_i) = f(w_i^t x) = w_i^t x \tag{3}$$

This rule can be considered a special case of delta learning rule.

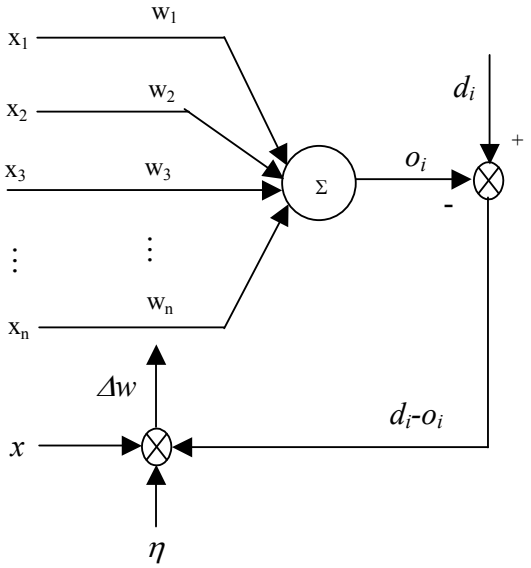
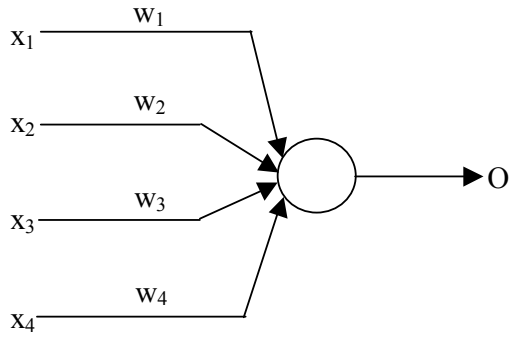


Figure 3: Widrow-Hoff learning

Example 1: Assume the neural network with a single neuron shown in Figure below having the initial weight vector w^1, and three input vectors as



$$w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix},\ x^1 = \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix},\ x^2 = \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} \text{ and } x^3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

The network needs to be trained with a learning constant $c=1$. The activation function is defined as $f(net) = net$. What will be the weight vector after 2nd iteration of Widrow-Hoff learning?

Solution:

Gradient Descent Rule

This rule is similar to the Delta rule in that the derivative of the transfer function is used to modify the error before it is applied to the connection weights. This rule is commonly used, even though the convergence to a stable point is very slowly. It has been shown that different learning rates for different layers of network help the learning process converge faster. Among the gradient descent-learning algorithms, backpropagation (BP) algorithm is most popular algorithm and it is an extension of the perceptrons to multi-layered neural network. There are a number of variations on the basic algorithm that are based on the other optimization techniques, such as conjugate gradient and Newton methods.

Delta learning rule

Delta learning rule is applicable for supervised training of neural networks and valid for continuous activation function. It minimises the squared error between the desired output y_d and actual output. Calculating the gradient vector with respect to w_i of the squared error defined as

$$E = \frac{1}{2}(y_d - o_i)^2 = \frac{1}{2}e^2 \tag{4}$$

where the output o_i is defined as

$$o_i = f(net_i) = f(w_i^t x) = w_i^t x \tag{5}$$

$f(.)$ is the activation function which is continuous for delta learning rule. The minimisation of the error requires the weight vector changes to be in the negative gradient direction, so we take

$$\Delta w_i = -\eta \nabla E \tag{6}$$

where η is the learning rate (constant). ∇E is defined as

$$\nabla E = -(y_d - o_i) f'(net) x \tag{7}$$

$f'(.)$ is the derivative of the activation function. The components of the gradient vectors are

$$\frac{\partial E}{\partial w_{ij}} = -(y_d - o_i) f'(net) x_j, \qquad j = 1,2,\cdots,n \tag{8}$$

From equations (6) and (7), the weight update for delta learning rule becomes

$$\Delta w_i = \eta (y_d - o_i) f'(net_i) x = \eta e_i f'(net) x \tag{9}$$

The process of delta learning rule is shown in Figure 4.

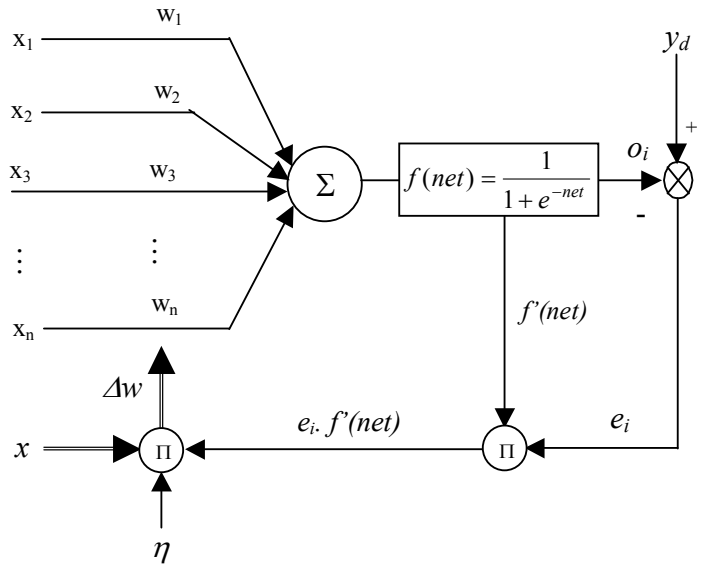
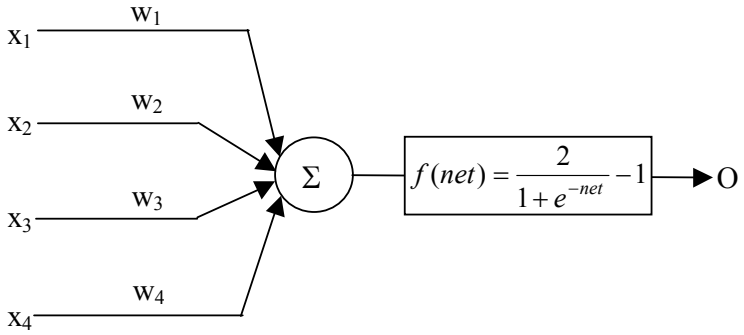


Figure 4: Delta learning rule

Example 2: Assume the neural network with a single neuron shown in Figure below having the initial weight vector w^1, and three input vectors as



$$w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix},\ x^1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix},\ x^2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \text{ and } x^3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

The desired responses for x^1, x^2, and x^3 are $d^1=-1$, $d^2=-1$ and $d^3=1$ respectively. The network needs to be trained with a learning constant $c=1$. The activation function is defined as

$$f(net) = \frac{2}{1+e^{-net}} - 1.$$

The delta-learning rule requires the value of $f'(.)$ to be computed in each step. For this purpose the following derivative is given as

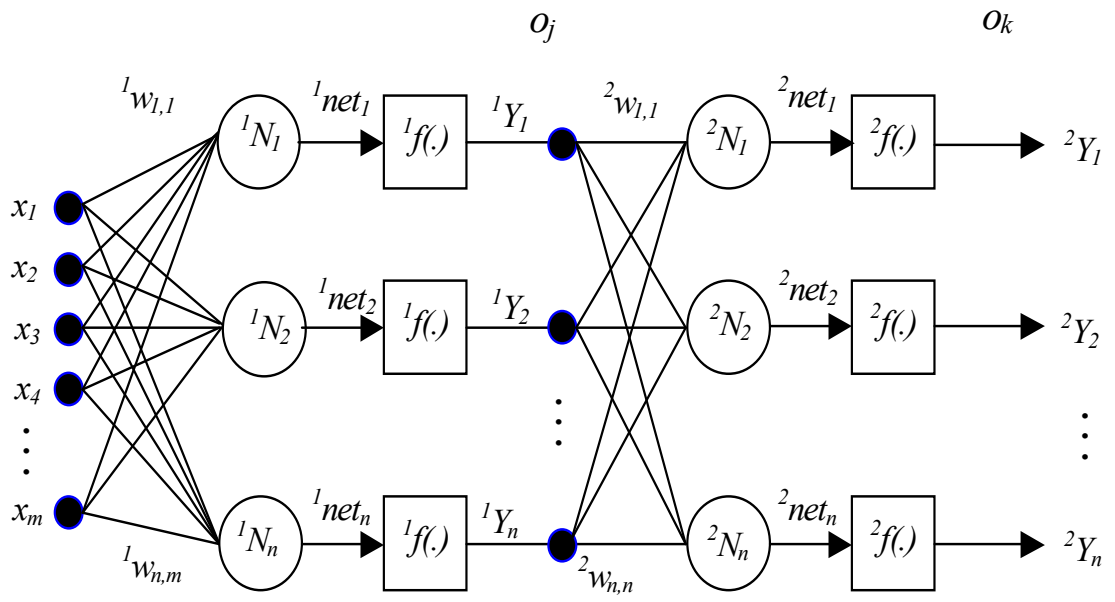
$$f'(net) = \frac{1}{2}(1 - o^2)$$

What will be the weight vector after 2nd iteration of Delta learning?

Solution:

Generalised Delta Learning Rule

Delta learning rule can be generalised and applied to any feedforward layered network. The architecture of a two-layer network is considered in this case and shown in Figure 5.



$$^1Y = {}^1f(^1W \cdot X + {}^1b) \qquad ^2Y = {}^2f(^2W \cdot {}^1Y + {}^2b)$$

Figure 5: Two-layer feedforward network.

The delta-learning rule can now be applied to adjust hidden layer weights (^1W) and output layer weights (^2W) of the two-layered network. This generalised delta-learning rule is error backpropagation learning – the algorithm is shown as a block-diagram in Figure 6.

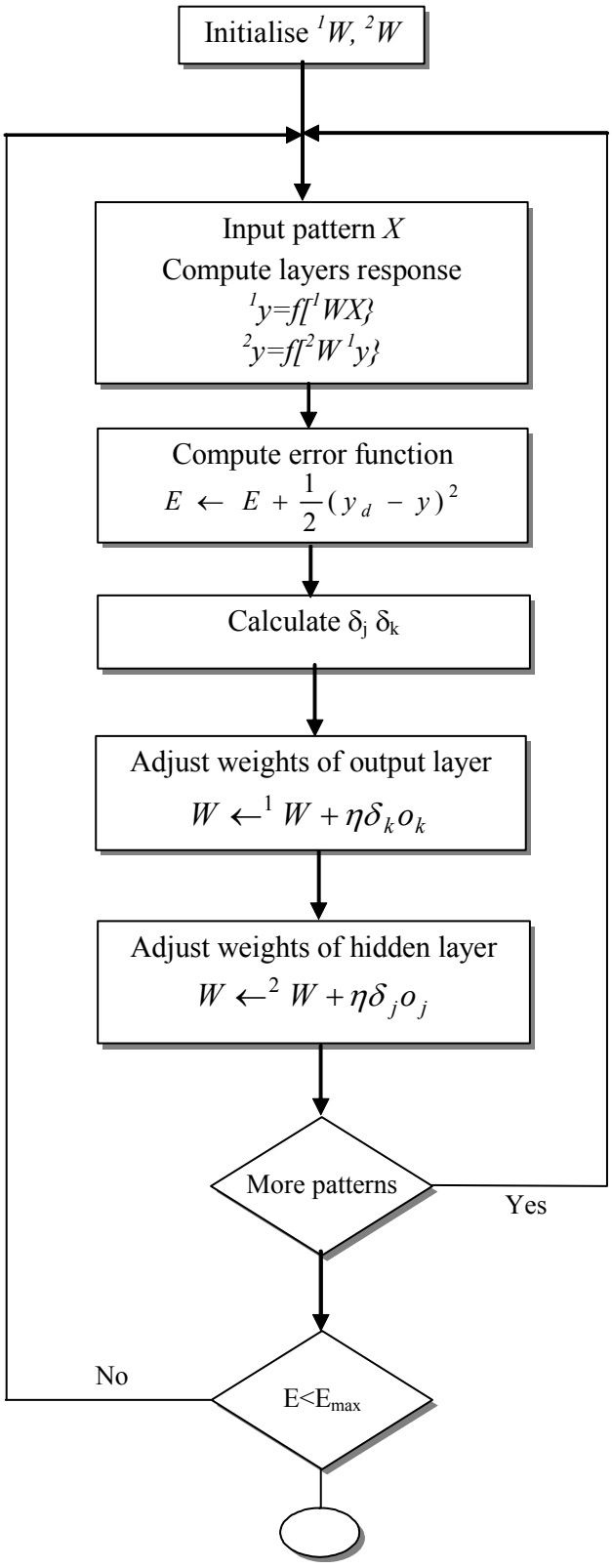


Figure 6: Error backpropagation rule.

Backpropagation learning algorithm

Standard backpropagation is a gradient descent algorithm. The term backpropagation refers to the manner in which gradient is computed for nonlinear multiplayer networks. For an example, we consider a three-layered NN shown in the Figure 7.

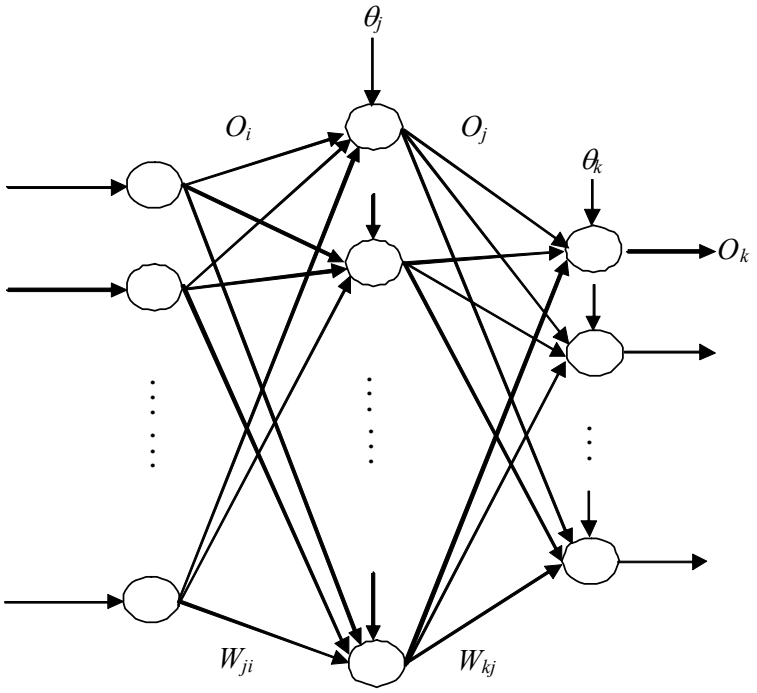


Figure 7: 3-layer network.

O_k, O_j, and $O_i \cong$ outputs of the output, hidden and input layers respectively
$w_{kj} \cong$ connection weight from hidden layer j to output layer k
$w_{ji} \cong$ connection weight from input layer i to hidden layer

Thus we have output layer

$$O_k = f(net_k) \tag{10}$$

$$net_k = \sum_j W_{kj} O_j + \theta_k \tag{11}$$

and the hidden layer

$$O_j = f(net_j) \tag{12}$$

$$net_j = \sum_i W_{ji} O_i + \theta_j \tag{13}$$

where *f(net)* is given by

$$f(net) = \frac{1}{1 + \exp(-net)} \tag{14}$$

The learning procedure involves the presentation of a set of pairs of input-output patterns. The net propagates the pattern inputs to outputs to produce its own output patterns and then compares this with the desired output. The difference is called error. If there is error, error is backpropagated to have weights and biases changed. If there is no error, learning stops.

Derivation of the backpropagation algorithms

Backpropagation algorithm is based on gradient descent method in that the error function is defined in (15). The principle of gradient descent method is shown in Figure 8, where the error function is compared with a rolling ball going down a valley.

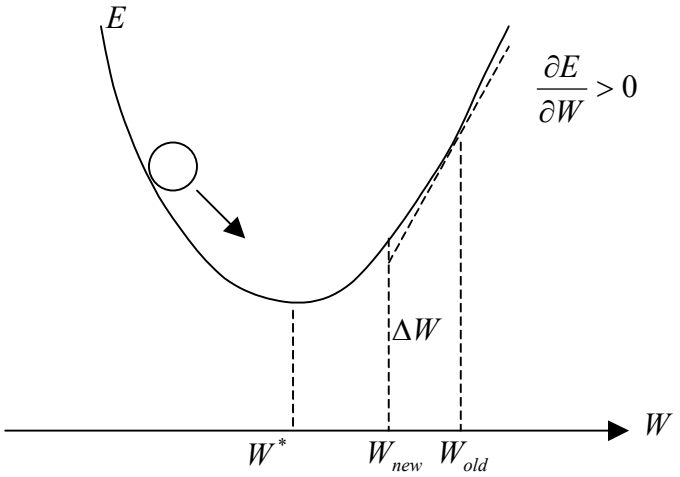


Figure 8: Principle of gradient descent method.

$$E = \frac{1}{2}\sum e^2 = \frac{1}{2}\sum_k \left(t_k - O_k\right)^2 \tag{15}$$

Calculation of the output layer weight-change: According to the steepest descent (gradient descent) method

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}} \tag{16}$$

where $\eta \hat{=}$ learning rate and $\eta>0$, $\Delta w_{kj} \hat{=}$ weight change and $\Delta w_{kj} = w_{kj}^{new} - w_{kj}^{old}$
Using chain rule, we get

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}} = -\delta_k \cdot \frac{\partial net_k}{\partial w_{kj}} \tag{17}$$

where $\delta_k = -\frac{\partial E}{\partial net_k}$ and termed as generalized error signal.

Now, we want to derive the term $\frac{\partial net_k}{\partial w_{kj}}$

$$\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial\left(\sum_j w_{kj} O_j + \theta_k\right)}{\partial w_{kj}} = O_j \tag{18}$$

To compute the term δ_k, we apply the chain rule

$$\delta_k = -\frac{\partial E}{\partial net_k} = -\frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial net_k} \tag{19}$$

$$\frac{\partial E}{\partial O_k} = \frac{\frac{1}{2}\sum_k (t_k - O_k)^2}{\partial O_k} = -(t_k - O_k) \tag{20}$$

$$\frac{\partial O_k}{\partial net_k} = f'(net_k) \tag{21}$$

Note: $f'(x)$ denotes the derivative of *f(x)* with respect to *x* and can be easily derived as follows

$$\begin{aligned} f'(x) &= \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{(1+e^{-x})}\left(1 - \frac{1}{1+e^{-x}}\right) \\ f'(x) &= f(x)(1 - f(x)) \end{aligned} \tag{22}$$

Hence,

$$\frac{\partial O_k}{\partial net_k} = O_k(1 - O_k) \tag{23}$$

Thus, we have

$$\begin{aligned} \Delta w_{kj} &= \eta \delta_k O_j \\ \delta_k &= O_k(1 - O_k)(t_k - O_k) \end{aligned} \tag{24}$$

Similarly, we can calculate bias change

$$\Delta\theta_k = -\eta\frac{\partial E}{\partial \theta_k} = \eta\left(-\frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial \theta_k}\right) = \eta\delta_k \frac{\partial\left(\sum_j w_{kj}O_j + \theta_k\right)}{\partial \theta_k} = \eta\delta_k \tag{25}$$

Calculation of the hidden layer weight-change: According to the gradient descent method

$$\Delta w_{ji} = -\eta\frac{\partial E}{\partial w_{ji}} \tag{26}$$

Using the chain rule we get

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial net_j}\cdot\frac{\partial net_j}{\partial w_{ji}} = -\delta_j \cdot \frac{\partial net_j}{\partial w_{ji}} \tag{27}$$

Now, we get

$$\frac{\partial net_j}{\partial w_{ji}} = O_i \tag{28}$$

Using chain rule, we get

$$\delta_j = -\frac{\partial E}{\partial net_j} = -\sum_k \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial O_j}\frac{O_j}{\partial net_j} \tag{29}$$

$$\delta_j = \sum_k \partial_k w_{kj} f'\left(net_j\right) \tag{30}$$

$$\delta_j = O_j\left(1-O_j\right)\sum_k \partial_k w_{kj} \tag{31}$$

Thus, we have

$$\Delta w_{ji} = \eta \delta_j O_i \tag{32}$$

$$\delta_j = O_j\left(1-O_j\right)\sum_k \partial_k w_{kj} \tag{33}$$

and the bias change

$$\Delta\theta_j = -\eta\frac{\partial E}{\partial \theta_j} = \ldots = \eta\delta_j \tag{34}$$

Backpropagation Algorithm

1. Initialise w_{ki}, w_{ji}, θ_k and θ_j and set learning rate η
2. Propagate inputs to network and calculate O_j, O_k
3. Calculate δ_k by the formula
$$\delta_k = O_k\left(1-O_k\right)\left(t_k - O_k\right)$$
4. Calculate change of weights and biases by
$$\Delta w_{kj} = \eta\delta_k O_j$$
$$\Delta\theta_k = \eta\delta_k$$
5. Calculate δ_j by the formula
$$\delta_j = O_j\left(1-O_j\right)\sum_k \delta_k w_{kj}$$
6. Calculate change of weights and biases by
$$\Delta w_{ji} = \eta\delta_j O_i$$
$$\Delta\theta_j = \eta\delta_j$$
7. Calculate new weights and biases
$$w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}$$
$$\theta_j(t+1) = \theta_j(t) + \Delta\theta_j$$
$$w_{kj}(t+1) = w_{kj}(t) + \Delta w_{kj}$$
$$\theta_k(t+1) = \theta_k(t) + \Delta\theta_k$$
8. set $t \leftarrow t+1$ and got to step 2.

Backpropagation algorithm can be illustrated with the following Figure

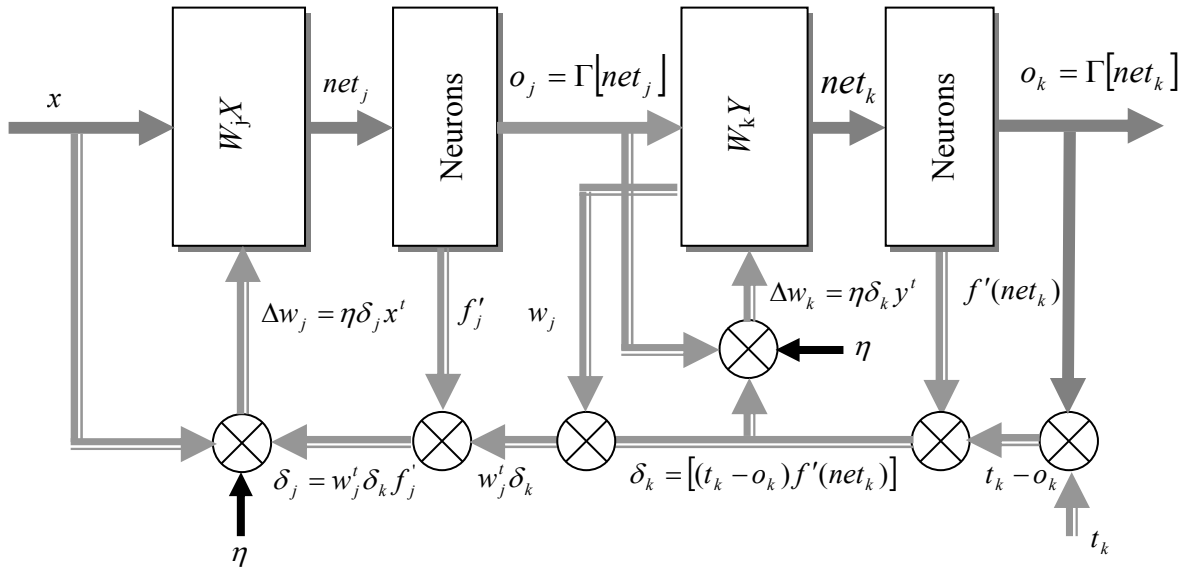


Figure 9: Backpropagation algorithm.

Problems with Backpropagation learning

BP is based on gradient descent algorithm to find the minimum of error. We seek global minima, which are sometimes surrounded by many local minima or plateau. Very often BP is stuck in a local minima or a plateau. This is explained in the Figure 9. A momentum term is added to the learning rule when BP is stuck in local minima. Likewise an acceleration term is added to learning rule when BP is stuck in a plateau. The modified learning rules are given by the equations below:

$$\Delta w_{kj}(t) = -\eta \frac{\partial E}{\partial w_{kj}} + \alpha \Delta w_{kj}(t-1) + \beta \Delta w_{kj}(t-2) \tag{35}$$

$$\Delta w_{ji}(t) = -\eta \frac{\partial E}{\partial w_{ji}} + \alpha \Delta w_{ji}(t-1) + \beta \Delta w_{ji}(t-2) \tag{36}$$

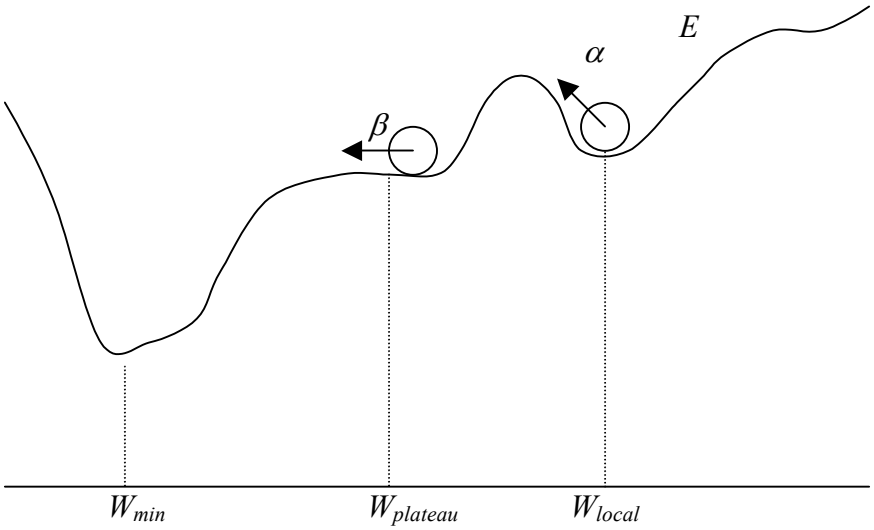


Figure 10: Momentum and acceleration.