INTRODUCTION TO MACHINE LEARNING

UNIT-V

Learning with Neural Networks

4.1 NEURON MODELS

The Neural network technology is discussed below:

4.1.1 Biological Neuron

- To the extent a human brain is understood today, it seems to operate as follows: bundles of neurons,or nerve fibers, form nerve structures.
- There are many different types of neurons in the nerve structure, each having a particular shape, size and length depending upon its function and utility in the nervous system.
- While each type of neuron has its own unique features needed for specific purposes, all neurons have two important structural components in common. These may be seen in the typical biological neuron shown in Fig.1.
- At one end of the neuron are a multitude of tiny, filament-like appendages called dendrites, which come together to form larger branches and trunks where they attach to soma, the body of the nerve cell.
- At the other end of the neuron is a single filament leading out of the soma, called an axon, which has extensive branching on its far end.
- These two structures have special electrophysiological properties which are basic to the function of neurons as information processors.

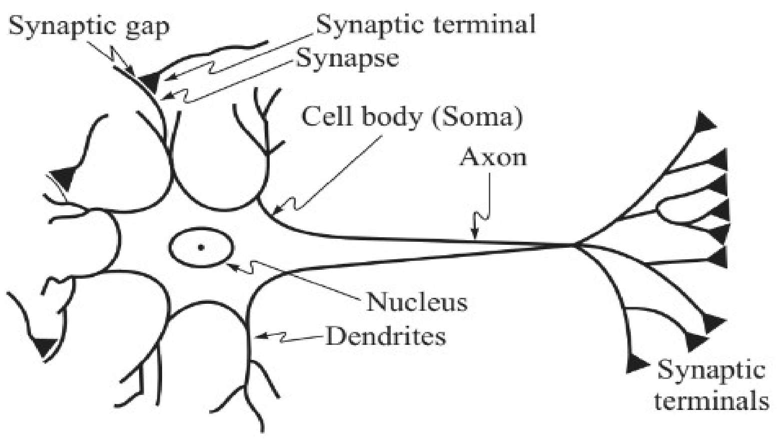


Figure 5.1 A typical biological neuron

Fig 1: A typical biological neuron

- Neurons are connected to each other via their axons and dendrites. Signals are sent through the axon of one neuron to the dendrites of other neurons. Hence, dendrites may be represented as the inputs to the neuron, and the axon as its output.
- Each neuron has many inputs through its multiple dendrites, whereas it has only one output through its single axon. The axon of each neuron forms connections with the dendrites of many other neurons, with each branch of the axon meeting exactly one dendrite of another cell at what is called a synapse.
- Actually, the axon terminals do not quite touch the dendrites of the other neurons, but are separated by a very small distance of between 50 and 200 angstroms. This separation is called the synaptic gap.
- On the other hand, the brain is composed of ten billion or so neurons. Each nerve cell can interact directly with up to 200,000 other neurons (though 1000 to 10,000) is typical
- The brain is organized into different regions, each responsible for different functions. The largest parts of the brain are the cerebral hemispheres, which occupy most of the interior of the skull.
- They are layered structures; the most complex being the outer layer, known as the cerebral cortex, where the nerve cells are extremely densely packed to allow greater interconnectivity. Interaction with the environment is through the visual, auditory and motion control (muscles and glands) parts of the cortex.
- A synapse pairs the axon with another cell's dendrite. It discharges chemicals known as neurotransmitters, when its potential is increased enough by the axon potential. The triggering of the synapse may require the arrival of more than one spike.
- The neurotransmitters emitted by the synapse diffuse across the gap, chemically activating gates on the dendrites, which, on opening, permit the flow of charged ions. This flow of ions, changes the dendritic potential and generates voltage pulse on the dendrite, which is then conducted into the neuron body.

4.1.2 Artificial Neuron

Artificial neurons bear only a modest resemblance to real things. They model approximately three of the processes that biological neurons perform (there are at least 150 processes performed by neurons in the human brain).

An artificial neuron
(i) evaluates the input signals, determining the strength of each one;
(ii) calculates a total for the combined input signals and compares that total to some threshold level; and
(iii) determines what the output should be.

i. Input and Outputs

Just as there are many inputs (stimulation levels) to a biological neuron, there should be many input signals to our artificial neuron (AN). All of them should come to our AN simultaneously.

In response, a biological neuron either 'fires' or 'doesn't fire' depending upon some threshold level. Our AN will be allowed a single output signal, just as is present in a biological neuron: many inputs, one output (Fig. 2).

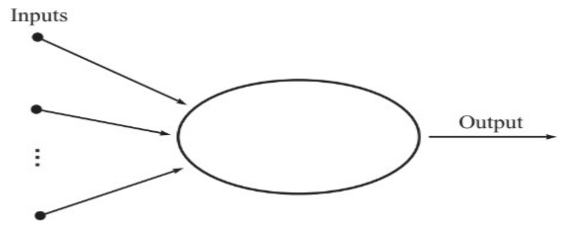


Figure 5.2 Many inputs, one output model of a neuron

Fig 2. Many inputs, one output model of a neuron

ii. Weighting Factors

Each input will be given a relative weighting, which will affect the impact of that input (Fig.3). This is something like varying synaptic strengths of the biological neurons—some inputs are more important than others in the way they combine to produce an impulse.
Weights are adaptive coefficients within the network, that determine the intensity of the input signal. In fact, this adaptability of connection strength is precisely what provides neural networks their ability to learn and store information, and, consequently, is an essential element of all neuron models.

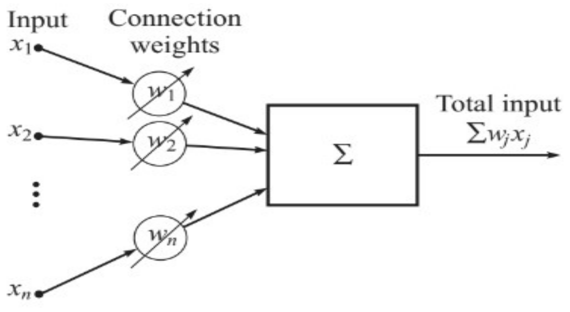


Figure 5.3 A neuron with weighted inputs

Fig 3. A neuron with weighted inputs

- Excitatory and inhibitory inputs are represented simply by positive or negative connection weights, respectively. Positive inputs promote the firing of the neuron, while negative inputs tend to keep the neuron from firing. Mathematically, we could look at the inputs and the weights on the inputs as vectors.

The input vector

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

and the connection weight vector

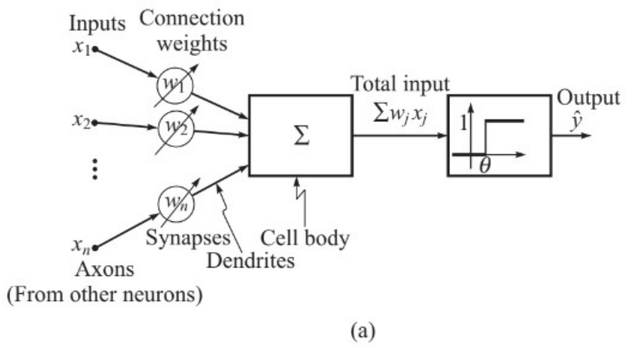
$$\mathbf{w}^T = [w_1\ w_2\ \dots\ w_n]$$

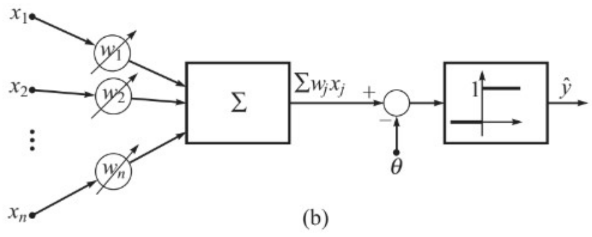
The total input signal is the product of these vectors. The result is a scalar

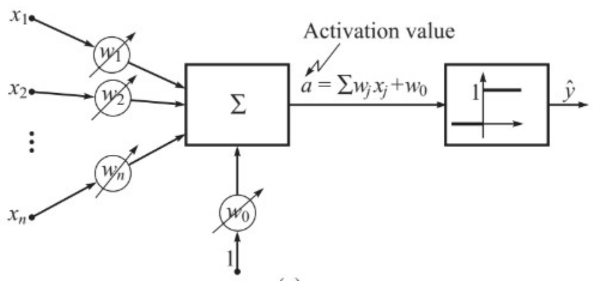
$$\sum_{j=1}^{N} w_j x_j = \mathbf{w}^T \mathbf{x}$$

iii. Activation Functions

- Artificial neurons use an activation function, often called a transfer function, to compute their activation as a function of total input stimulus.
- Figure 4a shows this neuron model. In this diagram, the neuron has been represented in such away that the correspondence of each element with its biological counterpart may be easily seen.
- Equivalently, the threshold value can be subtracted from the weighted sum and the resulting value compared to zero; if the result is positive, then output a 1, else output a 0. This is shown in Fig. 4b; note that the shape of the function is the same but now the jump occurs at zero.
- The threshold effectively adds an offset to the weighted sum. An alternative way of achieving the same effect is to take the threshold out of the body of the model neuron, and connect it to an extra input value that is fixed to be 'on' all the time. In this case, rather than subtracting the threshold value from the weighted sum, the extra input of +1 is multiplied by a weight and added in a manner similar to other inputs—this is known as biasing theneuron. Figure 4c shows a neuron model with a bias term. Note that we have taken constant input '1' with an adaptive weight 'w0' in our model.
- The first formal definition of a synthetic neuron model, based on the highly simplified considerations of the biological neuron, was formulated by McCulloch and Pitts (1943). The two-port model (inputs—activation value—output mapping) of Fig.4 is essentially the MP neuron model.







(c)

Fig 4. The MP neuron model

The MP artificial neuron model involves two important processes:

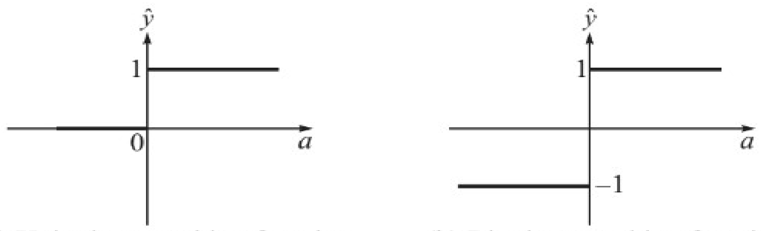
(i) Forming net activation by combining inputs. The input values are amalgamated by a weighted additive process to achieve the neuron activation value a (refer to Fig.4c).

(ii) Mapping this activation value a into the neuron output yˆ . This mapping from activation to output may be characterized by an 'activation' or 'squashing' function.

- For the activation functions that implement input-to-output compression or squashing, the range of the function is less than that of the domain. There is some physical basis for this desirable characteristic. Recall that in a biological neuron, there is a limited range of output (spiking frequencies).

- In the MP model, where DC levels replace frequencies, the squashing function serves to limit the output range. The squashing function shown in Fig. 5a limits the output values to {0,1}, while that in Fig.5b limits the output values to {–1, 1}. The activation function of

Fig. 5a is called unipolar, while that in Fig. 5b is called bipolar (both positive and negative responses of neurons are produced).



(a) Unipolar squashing function (b) Bipolar squashing function

Figure 5.5

Fig 5

4.1.3 Mathematical Model

A neuron model (a processing element/a unit/a node/a cell of our neural network), will be represented as follows:

The input vector

$$\mathbf{x} = [x_1 \; x_2 \; \dots \; x_n]^T;$$

the connection weight vector

$$\mathbf{w}^T = [w_1 \; w_2 \; \dots \; w_n];$$

the unity-input weight w_0 (bias term), and the output yˆ of the neuron are related by the following equation:

$$\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + w_0) = \sigma\left(\sum_{j=1}^{n} w_j x_j + w_0\right)$$

where σ () is the activation function (transfer function) of the neuron.

The weights are always adaptive. We can simplify our diagram as in Fig. 6a; adaptation need not be specifically shown in the diagram.

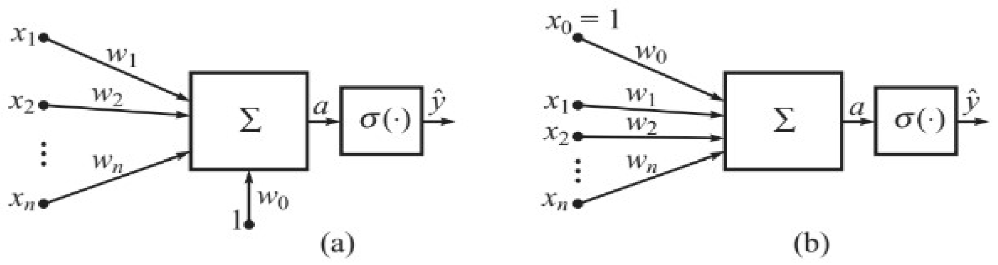


Figure 5.6 Mathematical model of a neuron (perceptron)

Fig 6. Mathematical model of a neuron (perceptron)

The bias term may be absorbed in the input vector itself as shown in Fig. .6b.

$$\begin{aligned}\hat{y} &= \sigma(a)\\ &= \sigma\left(\sum_{j=0}^{n} w_j x_j\right);\ x_0 = 1\\ &= \sigma\left(\sum_{j=1}^{n} w_j x_j + w_0\right) = \sigma(\mathbf{w}^T\mathbf{x} + w_0)\end{aligned}$$

In the literature, this model of an artificial neuron is also referred to as a perceptron (the name was given by Rosenblatt in 1958).

The expressions for the neuron output yˆ are referred to as the **cell recall mechanism**. They describe how the output is reconstructed from the input signals and the values of the cell parameters.

For many training algorithms,the derivative of the activation function is needed; therefore, the activation function selected must be differentiable. The logistic or sigmoid function, which satisfies this requirement, is the most commonly used soft-limiting activation function. The sigmoid function (Fig.7a):

$$\sigma(a) = \frac{1}{1 + e^{-\lambda a}}$$

is continuous and varies monotonically from 0 to 1 as a varies from -∞ to +∞. The gain of the sigmoid, λ , determines the steepness of the transition region.

4.2 NETWORK ARCHITECTURES

- In the biological brain, a huge number of neurons are interconnected to form the network and perform advanced intelligent activities.

- The artificial neural network is built by neuron models.Many different types of artificial neural networks have been proposed, just as there are many theories on how biological neural processing works.
- We may classify the organization of the neural networks largely into two types: a feedforward net and a recurrent net. The feedforward net has a hierarchical structure that consists of several layers, without interconnection between neurons in each layer, and signals flow from input to output layer in one direction. In the recurrent net, multiple neurons in a layer are interconnected to organize the network. In the following, we give typical characteristics of the feedforward net and the recurrent net, respectively.

4.2.1 Single-layer Networks

A one-layer network with n inputs and M neurons is shown below figure. . In the network, each input x_j ; j = 1, 2, …, n is connected to the qth neuron input through the weight w_{qj}; q = 1, 2, …, M. The qth neuron has a summer that gathers its weighted inputs to form its own scalar output

$$\sum_{j=1}^{n} w_{qj}\, x_j + w_{q0};\ q = 1, 2, \ldots, M$$

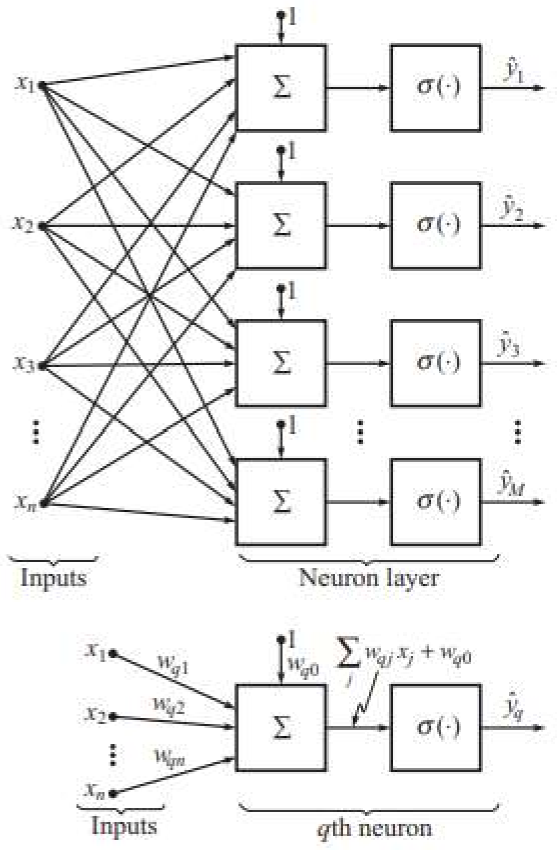


Fig. A one-layer network

Finally, the qth neuron outputs $y\hat{}_q$ through its activation function σ (.) is given by

$$\hat{y}_q = \sigma\left(\sum_{j=1}^{n} w_{qj}\, x_j + w_{q0}\right); \; q = 1, 2, \ldots, M$$
$$= \sigma(\mathbf{w}_q^T \mathbf{x} + w_{q0}); \; q = 1, 2, \ldots, M$$

where weight vector $\mathbf{w}_q$ is defined as,

$$\mathbf{w}_q^T = [w_{q1}\; w_{q2} \;\ldots\; w_{qn}]$$

4.2.2 Multilayer Networks

Neural networks normally have at least two layers of neurons, with the first layer neurons having nonlinear and differentiable activation functions. Such networks, as we will see, can approximate any nonlinear function. In real life, we are faced with nonlinear problems, and multilayer neural network structures have the capability of providing solutions to these problems.

Below figure shows a two-layer NN, with n inputs and two layers of neurons. The first of these layers has m neurons feeding into the second layer possessing M neurons. The first layer or the hidden layer, has m hidden-layer neurons; the second or the output layer, has M output-layer neurons. It is not uncommon for different layers to have different numbers of neurons. The outputs of the hidden layer are inputs to the following layer (output layer); and the network is fully connected. Neural networks possessing several layers are known as Multi-Layer Perceptrons (MLP); their computing power is meaningfully improved over the one-layer NN.

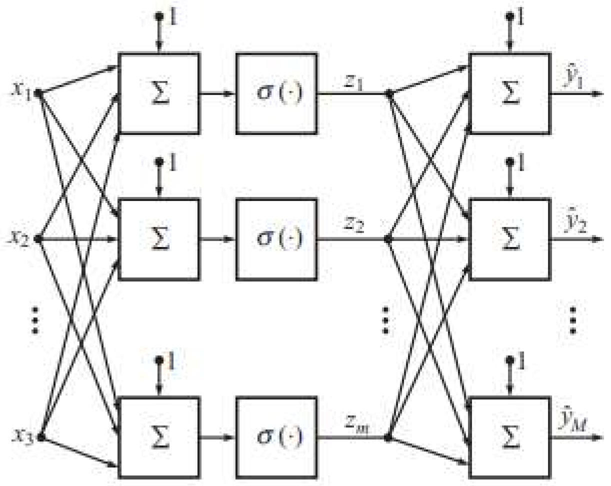


Fig. A two-layer network with one hidden layer

Neural networks possessing one or more hidden layers with sigmoidal/hyberbolic tangent hidden units are known as Multi-layer Perceptron (MLP) networks.

Defining the input terminals as x_j ; $j = 1, \ldots, n$; and the hidden-layer outputs as z_l , allows one to write

$$z_l = \sigma\left(\sum_{j=1}^{n} w_{lj} x_j + w_{l0}\right); l = 1, 2, \ldots, m$$

$$= \sigma(\mathbf{w}_l^T \mathbf{x} + w_{l0})$$

where

$$\mathbf{w}_l^T \triangleq [w_{l1}\ w_{l2} \ldots w_{ln}]$$

are the weights connecting input terminals to hidden layer.

Defining the output-layer nodes as $\hat{y}_q$, one may write the *NN* output as,

$$\hat{y}_q = \left(\sum_{l=1}^{m} v_{ql} z_l + v_{q0}\right); q = 1, \ldots, M$$

$$= \mathbf{v}_q^T \mathbf{z} + v_{q0}$$

where

$$\mathbf{v}_q^T \triangleq [v_{q1}\ v_{q2} \cdots v_{qm}]$$

are the weights connecting hidden layer to output layer.

4.3 PERCEPTRONS

One type of ANN system is based on a unit called a perceptron, illustrated in below figure A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise.

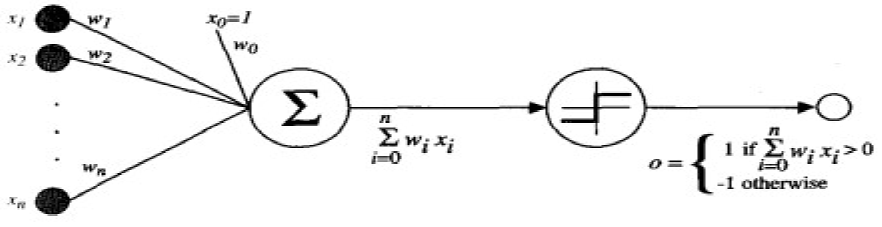

x_1
w_1
x_2
w_2
$x_0=1$
w_0
w_n
x_n
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

FIGURE 4.2
A perceptron.

Fig: A perceptron

More precisely, given inputs xl through x,, the output o(x1, . . . , x,) computed by the perceptron is

$$o(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

where each w_i is a real-valued constant, or weight, that determines the contribution of input x_i to the perceptron output. Notice the quantity $(-w_O)$ is a threshold that the weighted combination of inputs $w_lx_l + \ldots + w_nx_n$ must surpass in order for the perceptron to output a 1. Simplified notation is

$$o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$$

Where

$$sgn(y) = \begin{cases} 1 \text{ if } y > 0 \\ -1 \text{ otherwise} \end{cases}$$

4.3.1 Representation Power of Perceptrons

We can view the perceptron as representing a hyperplane decision surface in the n-dimensional space of instances (i.e., points). The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side, as illustrated in below figure.

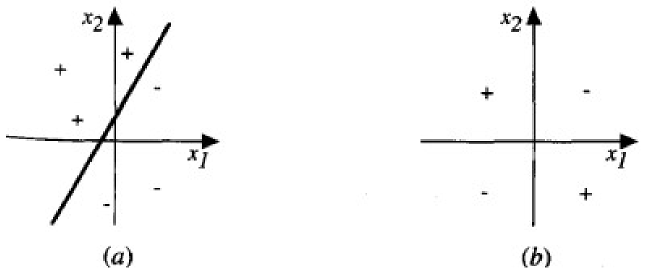


Fig: hyperplane

4.3.2 The Perceptron Training Rule

The perceptron rule and the delta are guaranteed to converge to somewhat different acceptable hypotheses, under somewhat different conditions. They are important to ANNs because they provide the basis for learning networks of many units.

One way to learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example. This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly. Weights are modified at each step according to the perceptron training rule, which revises the weight wi associated with input xi according to the rule.

$$w_i \leftarrow w_i + \Delta w_i$$

Where

$$\Delta w_i = \eta(t - o)x_i$$

Here t is the target output for the current training example, o is the output generated by the perceptron, and q is a positive constant called the learning rate. The role of the learning rate is to moderate the degree to which weights are changed at each step. It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases.

4.4 THE ERROR CORRECTION DELTA RULE

The gradient descent strategy for adapting weights for a single neuron having differentiable activation function is demonstrated. A neural unit with any differentiable function s(a) is shown in below figure. It first computes a linear combination of its inputs (activation value a); then applies nonlinear activation functions (a)Learning with Neural Networks (NN) to the result. The output yˆ of nonlinear unit is a continuous function of its input a. More precisely, the nonlinear unit computes its output as,

$$\hat{y} = \sigma(a)$$

$$a = \sum_{j=1}^{n} w_j x_j + w_0$$

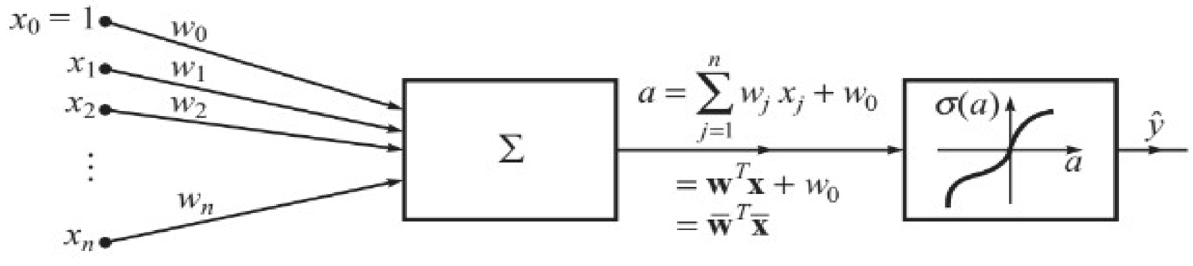

$x_0 = 1$
x_1
x_2
x_n
w_0
w_1
w_2
w_n
Σ
$a = \sum_{j=1}^{n} w_j x_j + w_0$
$= \mathbf{w}^T\mathbf{x} + w_0$
$= \bar{\mathbf{w}}^T\bar{\mathbf{x}}$
$\sigma(a)$
a
$\hat{y}$

Figure 5.13 Neural unit with any differentiable activation function

Fig. Neural unit with any differentiable activation function

The problem is to find the expression for the learning rule for adapting weights using a trainingset of pairs of input and output patterns; the learning is in stochastic gradient descent mode, as in the last section. We begin by defining error function E(k):

$$E(k) = \tfrac{1}{2}(y^{(i)} - \hat{y}(k))^2 = \tfrac{1}{2}[e(k)]^2$$

$$e(k) = y^{(i)} - \hat{y}(k)$$

$$\hat{y}(k) = \sigma\left(\sum_{j=1}^{n} w_j(k)\, x_j^{(i)} + w_0(k)\right)$$

For each training example i, weights w_j; j = 1, ..., n (and bias w_0) are updated by adding to it **δ**wj (and δw0).

$$\Delta w_j(k) = -\eta \frac{\partial E(k)}{\partial w_j(k)}$$

$$w_j(k+1) = w_j(k) - \eta \frac{\partial E(k)}{\partial w_j(k)}$$

$$w_0(k+1) = w_0(k) - \eta \frac{\partial E(k)}{\partial w_0(k)}$$

The E(k) is a nonlinear function of the weights now, and the gradient cannot be calculated following the equations derived in the last section for a linear neuron. Fortunately, the calculation of the gradient is straight forward in the nonlinear case as well. For this purpose, the chain rule is,

$$\frac{\partial E(k)}{\partial w_j(k)} = \frac{\partial E(k)}{\partial a(k)} \frac{\partial a(k)}{\partial w_j(k)}$$

where the first term on the right-hand side is a measure of an error change due to the activation value a(k) at the kth iteration, and the second term shows the influence of the weights on that particular activation value a(k). Applying the chain rule again, we get,

$$\frac{\partial E(k)}{\partial w_j(k)} = \frac{\partial E(k)}{\partial e(k)} \frac{\partial e(k)}{\partial \hat{y}(k)} \frac{\partial \hat{y}(k)}{\partial a(k)} \frac{\partial a(k)}{\partial w_j(k)}$$

$$= e(k)\,[-1]\,\frac{\partial \sigma(a(k))}{\partial a(k)}\, x_j^{(i)}$$

$$= -\, e(k)\, \sigma'(a(k))\, x_j^{(i)}$$

The learning rule can be written as,

$$w_j(k+1) = w_j(k) + \eta\, e(k)\, \sigma'(a(k))\, x_j^{(i)}$$
$$w_0(k+1) = w_0(k) + \eta\, e(k)\, \sigma'(a(k))$$

This is the most general learning rule that is valid for a single neuron having any nonlinear and differentiable activation function and whose input is formed as a product of the pattern and weight vectors. It follows the LMS algorithm for a linear neuron presented in the last section, which was an early powerful strategy for adapting weights using data pairs only.

This rule is also known as delta learning rule with delta defined as,

$$\begin{aligned}\delta(k) &= e(k)\, \sigma'(a(k)) \\ &= (y^{(i)} - \hat{y}(k))\, \sigma'(a(k))\end{aligned}$$

In terms of δ(k), the weights-update equations become

$$w_j(k+1) = w_j(k) + \eta\, \delta(k)\, x_j^{(i)}$$
$$w_0(k+1) = w_0(k) + \eta\, \delta(k)$$

It should be carefully noted that the δ(k) in these equations is not the error but the error change $-\frac{\partial E(k)}{\partial a(k)}$ due to the input a(k) to the nonlinear activation function at the kth iteration:

$$\begin{aligned}-\frac{\partial E(k)}{\partial a(k)} &= -\frac{\partial E(k)}{\partial e(k)}\frac{\partial e(k)}{\partial \hat{y}(k)}\frac{\partial \hat{y}(k)}{\partial a(k)} \\ &= -\, e(k)\, [-1]\, \sigma'(a(k)) \\ &= e(k)\, \sigma'(a(k)) = \delta(k)\end{aligned}$$

Thus, δ(k) will generally not be equal to the error e(k). We will use the term error signal for δ (k), keeping in mind that, in fact, it represents the error change.

In the world of neural computing, the error signal δ(k) is of highest importance. After a hiatusin the development of learning rules for multilayer networks for about 20 years, the adaptation rule based on delta rule made a breakthrough in 1986 and was named the generalized delta learning rule. Today, the rule is also known as the error backpropagation learning rule (discussed in the next section).

Interestingly, for a linear activation function

$$\sigma(a(k)) = a(k)$$

Therefore,

$$\sigma'(a(k)) = 1$$

and

$$\delta(k) = e(k)\ \sigma'(a(k)) = e(k)$$

That is, delta represents the error itself. Therefore, the delta rule for a linear neuron is same as the LMS learning rule presented in the previous section.

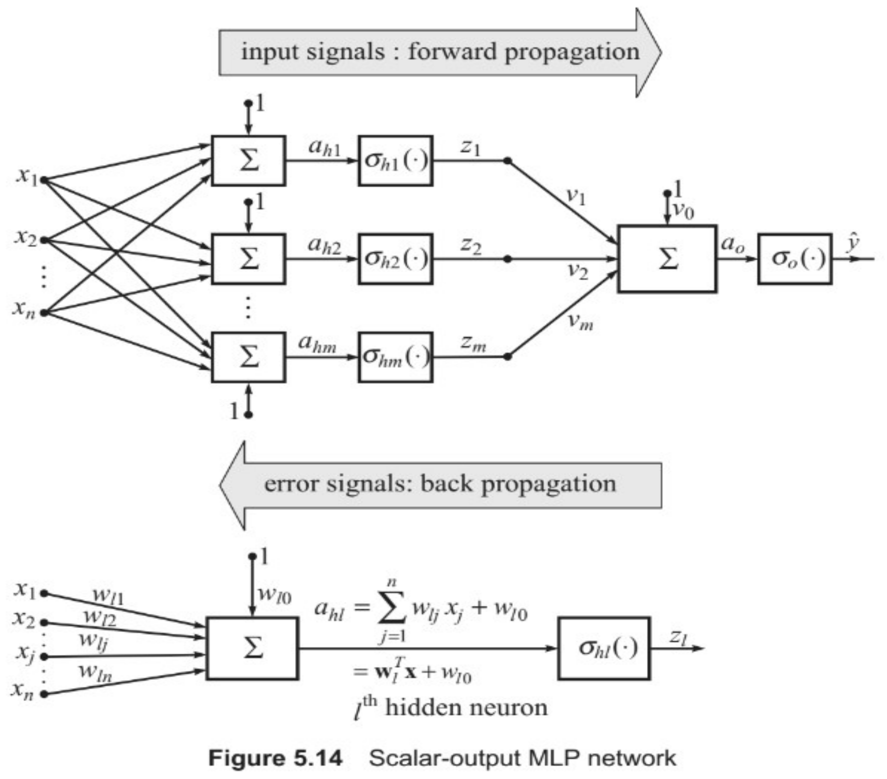
4.5 MULTI-LAYER PERCEPTRON (MLP) NETWORKS AND THE ERROR-BACKPROPAGATION ALGORITHM

- Single (hard-limiting) perceptron can only express linear decision surfaces, and single linear neuron can only approximate linear functions.
- In contrast, MLP networks trained by the backpropagation algorithm are capable of expressing a rich variety of nonlinear decision surfaces/approximating nonlinear functions. This section discusses how to learn such MLP networks using gradient descent algorithms similar to the ones described in previous sections.
- The backpropagation algorithm learns the weights for an MLP network, given a network with a fixed set of units and interconnections. It employs gradient descent to attempt to minimize the squared error between the network outputs and the target values for these outputs.
- A typical feedforward neural network is made up of a hierarchy of layers, and the neurons in the network are arranged along these layers. The external environment is connected to the network through input terminals, and the output-layer neurons.
- To build an artificial neural network, we must first decide how many layers of neurons are to be used and how many neurons in each layer. In other words, we must first choose the network architecture.
- The number of input terminals, and the number of output nodes in output layer depend on the nature of the data presented to the network, and the type of the output desired from it, respectively.
- A multi-layer perceptron (MLP) network is a feedforward neural network with one or more hidden layers. Each hidden layer has its own specific function. Input terminals accept

input signals from the outside world and redistribute these signals to all neurons in a hidden layer.

- The output layer accepts a stimulus pattern from a hidden layer and establishes the output pattern of the entire network. Neurons in the hidden layers perform transformation of input attributes; the weights of the neurons represent the features in the transformed domain.
- These features are then used by the output layer in determining the output pattern. A hidden layer 'hides' its desired output. The training data provides the desired output of the network; that is, the desired outputs of output layer. There is no obvious way to know what the desired outputs of the hidden layers should be.

The derivation of error-backpropagation algorithm will be given here for two MLP structures shown in below figures.



1

Fig: Scalar-output MLP network

The functions $\sigma_o(.)/\sigma_{oq}(.)$ are the linear/log-sigmoid activation functions of the output layer, and the functions $\sigma_{hl}(.)$ are the activation functions of the hidden layer (log-sigmoid or tan-sigmoid). The structure of output layer is given below figure.

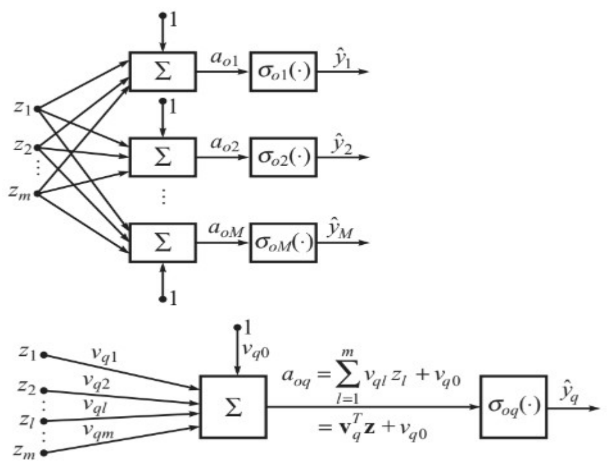


Figure 5.15 Output layer of vector-output MLP network

Fig: Output layer of vector-output MLP network

When x is fed to the input terminals (including the bias), the activation spreads in the feed forward direction, and the values z_l of the hidden units are computed (Each hidden unit is a perceptron on its own and applies the nonlinear sigmoid function to its weighted sum of inputs). The outputs $y\hat{}_q$ in the output layer are computed taking the hidden-layer outputs z_l as their inputs. Each output unit applies the log-sigmoid function to its weighted sum of inputs. It can be said that the hidden units make a nonlinear transformation from the n-dimensional input space to the m-dimensional space spanned by the hidden units, and in this space the output layer implements a log-sigmoid function.

4.5.1 The Generalized Delta Rule

The derivation here is of learning rule for the adaptation of weights in an incremental mode.

4.5.1.1 Update rule for Output-units Weights

We begin by defining cost function (sum-of-error-squares) for this neural network having M output-layer neurons. At each iteration step k,

$$E(k) = \tfrac{1}{2}\sum_{q=1}^{M}\left(y_q^{(i)} - \hat{y}_q(k)\right)^2 = \tfrac{1}{2}\sum_{q=1}^{M}[e_q(k)]^2$$

$$\hat{y}_q(k) = \sigma_{oq}\left(\sum_{l=1}^{m}[v_{ql}(k)\, z_l(k)] + v_{q0}(k)\right)$$

$$= \sigma_{oq}(a_{oq}(k))$$

From forward propagation phase, we have,

$$z_l(k) = \sigma_{hl}(a_{hl}(k))$$
$$= \sigma_{hl}\left(\sum_{j=1}^{n} [w_{lj}(k)\, x_j^{(i)}] + w_{l0}(k) \right)$$

We have used subscript 'o' for the output layer and subscript 'h' for the hidden layer. This is necessary because the error signal terms (the delta values) for output layer neurons must be distinguished from those for hidden layer processing units. The input a_{oq} to the qth output unit

$$a_{oq}(k) = \sum_{l=1}^{m} (v_{ql}(k)\, z_l(k)) + v_{q0}(k)$$

The error signal term for the qth neuron is defined as

$$\delta_{oq}(k) = -\, \frac{\partial E(k)}{\partial a_{oq}(k)}$$

Applying the chain rule, the gradient of the cost function with respect to the weight v_{ql} is,

$$\frac{\partial E(k)}{\partial v_{ql}(k)} = \frac{\partial E(k)}{\partial a_{oq}(k)} \frac{\partial a_{oq}(k)}{\partial v_{ql}(k)}$$
$$= -\, \delta_{oa}(k)\, z_l(k)$$

The weight change from can be written as,

$$\Delta v_{ql}(k) = -\, \eta\, \frac{\partial E(k)}{\partial v_{ql}(k)} = \eta\, \delta_{oq}(k)\, z_l(k)$$

Applying the chain rule, the expression for error signal is,

$$\delta_{oq}(k) = -\, \frac{\partial E(k)}{\partial a_{oq}(k)} = -\, \frac{\partial E(k)}{\partial \hat{y}_q(k)} \frac{\partial \hat{y}_q(k)}{\partial a_{oq}(k)}$$
$$= e_q(k)\, \sigma'_{oq}(a_{oq}(k))$$

where the term $\sigma'_{oq}(a_{oq}(k))$ represents the slope $\partial \hat{y}_q(k) / \partial a_{oq}(k)$ of the qth output neuron's activation function, assumed in general, to be any nonlinear differentiable function. For our specific case of log-sigmoid output units.

$$\sigma'_{oq}(a_{oq}(k)) = \frac{\partial \sigma_{oq}(a_{oq}(k))}{\partial a_{oq}(k)} = \hat{y}_q(k)\,[1 - \hat{y}_q(k)]$$

resulting in a simple expression of the error signal term:

$$\delta_{oq}(k) = e_q(k)\;\hat{y}_q(k)\,[1 - \hat{y}_q(k)]$$

Finally, the weight adjustments can be calculated from

$$v_{ql}(k+1) = v_{ql}(k) + \eta\;\delta_{oq}(k)\,z_l(k)$$
$$v_{q0}(k+1) = v_{q0}(k) + \eta\;\delta_{oq}(k)$$

4.5.1.2 Update-Rule for Hidden-Units Weights

The problem at this point is to calculate the error signal terms $\boldsymbol{\delta_{hl}}$ for the hidden-layer neurons, to update the weights w_{lj} between input terminals and hidden layer. The derivation of the expression for δ_{hl} was a major breakthrough in the learning procedure for multilayer neural networks.

The generalized delta rule provides a wayout for the computation of error signals terms for hidden units.

The generalized delta rule computes the error signal terms $\boldsymbol{\delta_{hl}}$ for hidden layer units by summing the output error signal terms **δ**oq for each output unit influenced by a_{hl}, weighting each of the $\boldsymbol{\delta_{oq}}$'s by v_{ql}—the weights from the hidden unit to output units. Thus, by error backpropagation from output layer to the hidden layer, we take into account the indirect ways in which w_{lj} can influence the network outputs, and hence the error E. The power of backpropagation is that it allows us to calculate an 'effective' error for each hidden node and, thus, derive the learning rule for input terminals to hidden layer weights

The derivation of the learning rule or of the equations for the weight change $\boldsymbol{\Delta} w_{lj}$ of any hiddenlayer neuron follows the gradient procedure used earlier for output-layer neurons:

$$\Delta w_{lj}(k) = -\eta\,\frac{\partial E(k)}{\partial w_{lj}(k)}$$

$$w_{lj}(k+1) = w_{lj}(k) - \eta \frac{\partial E(k)}{\partial w_{lj}(k)}$$

Similar equations hold for bias weights w_{l0}

Applying the chian rule, the gradient of the cost function with respect to the weight w_{lj} is

$$\frac{\partial E(k)}{\partial w_{lj}(k)} = \frac{\partial E(k)}{\partial a_{hl}(k)} \frac{\partial a_{hl}(k)}{\partial w_{lj}(k)}$$

Where input a_{hl} to each hidden-layer activation function is given as

$$a_{hl}(k) = \sum_{j=1}^{n} w_{lj}(k)\, x_j^{(i)} + w_{l0}(k)$$

The error signal term for lth neuron is given as

$$\delta_{hl}(k) = -\frac{\partial E(k)}{\partial a_{hl}(k)}$$

Therefore, the gradient of the cost function becomes

$$\frac{\partial E(k)}{\partial w_{lj}(k)} = \frac{\partial E(k)}{\partial a_{hl}(k)} \frac{\partial a_{hl}(k)}{\partial w_{lj}(k)}$$
$$= -\,\delta_{hl}(k)\, x_j^{(i)}$$

The weight-update equation takes the form

$$w_{lj}(k+1) = w_{lj}(k) + \eta\, \delta_{hl}(k)\, x_j^{(i)}$$

For the bias weights,

$$w_{l0}(k+1) = w_{l0}(k) + \eta\, \delta_{hl}(k)$$

Now the problem in hand is to calculate the error signal term **δ**hl for hidden-layer neuron in terms of error signal terms **δ**oq of the output-layer neurons employing error backpropagation.

The activation a_{hl} of lth hidden-layer neuron is given as,

$$a_{hl}(k) = \sum_{j=1}^{n} w_{lj}(k)\, x_j^{(i)} + w_{l0}(k)$$

The error signal term for lth neuron,

$$\delta_{hl}(k) = -\frac{\partial E(k)}{\partial a_{hl}(k)}$$

Since a_{hl} contributes to errors at all output-layer neurons, we have the chain rule

$$\frac{\partial E(k)}{\partial a_{hl}(k)} = \sum_{q=1}^{M} \frac{\partial E(k)}{\partial a_{oq}(k)} \frac{\partial a_{oq}(k)}{\partial a_{hl}(k)}$$

$$= \sum_{q=1}^{M} -\delta_{oq}(k) \frac{\partial a_{oq}(k)}{\partial a_{hl}(k)}$$

$$= \sum_{q=1}^{M} -\delta_{oq}(k) \frac{\partial a_{oq}(k)}{\partial z_l(k)} \frac{\partial z_l(k)}{\partial a_{hl}(k)}$$

Since,

$$a_{oq}(k) = \sum_{l=1}^{m} v_{ql}(k)\, z_l(k) + v_{q0}$$

we have,

$$-\delta_{hl}(k) = \frac{\partial E(k)}{\partial a_{hl}(k)} = \sum_{q=1}^{M} -\delta_{oq}(k)\, v_{ql}(k)\, \sigma'_{hl}(a_{hl})$$

Assuming unipolar sigmoid activation functions in hidden-layer neurons, we have,

$$\sigma'_{hl}(a_{hl}(k)) = z_l(k)\,[1 - z_l(k)]$$

This gives,

$$\delta_{hl}(k) = z_l(k)\,[1 - z_l(k)] \sum_{q=1}^{M} \delta_{oq}(k)\, v_{ql}(k)$$

***** SUMMARY OF BACK PROPAGATION ALGORITHM*****

Given a set of N data pairs $\{x^{(i)}, y^{(i)}\}$; i = 1, 2, …, N, that are used for training:

$x = [x_1\ x_2\ \dots\ x_n]^T = \{xj\}$; j = 1, 2, …, n

$y = [y_1\ y_2\ \dots\ y_M]^T = \{y_q\}$; q = 1, 2, …, M

Forward Recursion

Step 1: Choose the number of units m in the hidden layer, the learning rate h, and predefine the maximally allowed (desired) error E_{des}.

Step 2: Initialize weights w_{lj}, w_{l0}, v_{ql}, v_{q0}; l = 1, …, m.

Step 3: Present the input $x^{(i)}$ (drawn in sequence or randomly) to the network.

Step 4: Consequently, compute the output from the hidden and output layer neurons using Eqns (5.53a–5.53b).

$$z_l(k) = \sigma_{hl}\left(\sum_{j=1}^{n}(w_{lj}(k)\,x_j^{(i)}) + w_{l0}(k)\right);\ l = 1, \ldots, m$$

$$\hat{y}_q(k) = \sigma_{oq}\left(\sum_{l=1}^{m}(v_{ql}(k)\,z_l(k)) + v_{q0}(k))\right);\ q = 1, \ldots, M$$

with initial weights w_{lj}, w_{l0}, v_{ql}, v_{q0}, randomly chosen

Step 5: Find the value of the sum of error-squares cost function E(k) at the kth iteration for the data pair applied and given weights (in the first step of an epoch, initialize E(k) = 0):

$$E(k) \leftarrow \tfrac{1}{2}\sum_{q=1}^{M}(y_q^{(i)} - \hat{y}_q(k))^2 + E(k)$$

Note that the value of the cost function is accumulated over all the data pairs.

Backward Recursion

Step 6: Calculate the error signals δ_{oq} and δ_{hl} for the output layer neurons and hidden layer neurons, respectively, using Eqns (5.53c–5.53d).

$$\delta_{oq}(k) = [y_q^{(i)} - \hat{y}_q(k)]\ \hat{y}_q(k)[1 - \hat{y}_q(k)]$$

$$\delta_{hl}(k) = z_l(k)\,[1 - z_l(k)]\sum_{q=1}^{M}\delta_{oq}(k)\,v_{ql}(k)$$

Update Weights

Step 7: Calculate the updated output layer weights $v_{ql}(k + 1)$, $v_{q0}(k + 1)$ and updated hidden layer weights $w_{lj}(k + 1)$, $w_{l0}(k + 1)$ as

$$v_{ql}(k+1) = v_{ql}(k) + \eta\,\delta_{oq}(k)\,z_l(k)$$

$$v_{q0}(k+1) = v_{q0}(k) + \eta\,\delta_{oq}(k)$$

$$w_{lj}(k+1) = w_{lj}(k) + \eta\,\delta_{hl}(k)\,x_j^{(i)}$$

$$w_{l0}(k+1) = w_{l0}(k) + \eta\,\delta_{hl}(k)$$

Step 8: If i < N, go to step 3; otherwise go to step 9.

Step 9: The learning epoch (sweep through all data pairs) completed: i = N. For $E_N < E_{des}$, terminate training. Otherwise go to step 3 and start a new learning epoch.