Chapter 9

Neural networks

9.1 Introduction

An *Artificial Neural Network* (ANN) models the relationship between a set of input signals and an output signal using a model derived from our understanding of how a biological brain responds to stimuli from sensory inputs. Just as a brain uses a network of interconnected cells called neurons to create a massive parallel processor, ANN uses a network of artificial neurons or nodes to solve learning problems.

9.2 Biological motivation

Let us examine how a biological neuron functions. Figure 9.2 gives a schematic representation of the functioning of a biological neuron.

In the cell, the incoming signals are received by the cell's *dendrites* through a biochemical process. The process allows the impulse to be weighted according to its relative importance or frequency. As the cell body begins accumulating the incoming signals, a threshold is reached at which the cell fires and the output signal is transmitted via an electrochemical process down the *axon*. At the axon's terminals, the electric signal is again processed as a chemical signal to be passed to the neighboring neurons across a tiny gap known as a *synapse*.[1]

Biological learning systems are built of very complex webs of interconnected neurons. The human brain has an interconnected network of approximately 10^{11} neurons, each connected, on an average, to 10^4 other neurons. Even though the neuron switching speeds are much slower than than

[1] Neuron. (2018, February 15). In Wikipedia, The Free Encyclopedia. Retrieved 01:44, February 23, 2018.

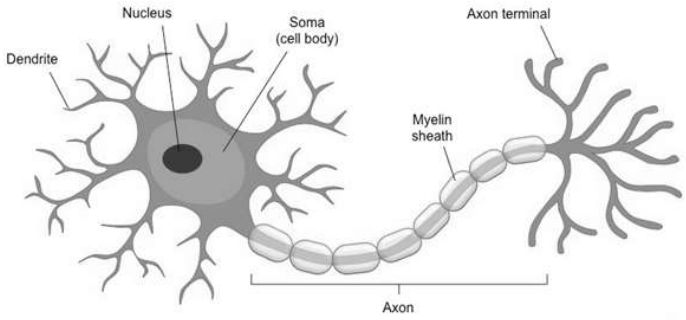


Figure 9.1: Anatomy of a neuron

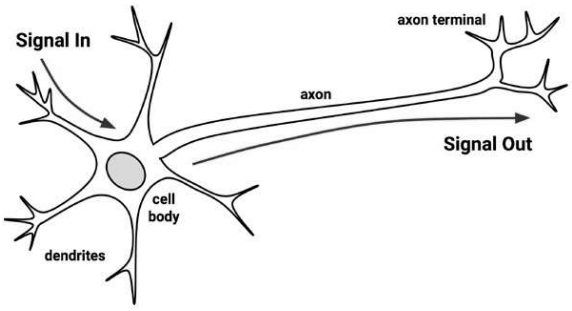


Figure 9.2: Flow of signals in a biological neuron

computer switching speeds, we are able to take complex decisions relatively quickly. Because of this, it is believed that the information processing capabilities of biological neural systems is a consequence of the ability of such systems to carry out a huge number of parallel processes distributed over many neurons. The developments in ANN systems are motivated by the desire to implement this kind of highly parallel computation using distributed representations.

9.3 Artificial neurons

Definition

An *artificial neuron* is a mathematical function conceived as a model of biological neurons. Artificial neurons are elementary units in an artificial neural network. The artificial neuron receives one or more inputs (representing excitatory postsynaptic potentials and inhibitory postsynaptic potentials at neural dendrites) and sums them to produce an output. Each input is separately weighted, and the sum is passed through a function known as an *activation function* or *transfer function*.

Schematic representation of an artificial neuron

The diagram shown in Figure **??** gives a schematic representation of a model of an artificial neuron. The notations in the diagram have the following meanings:

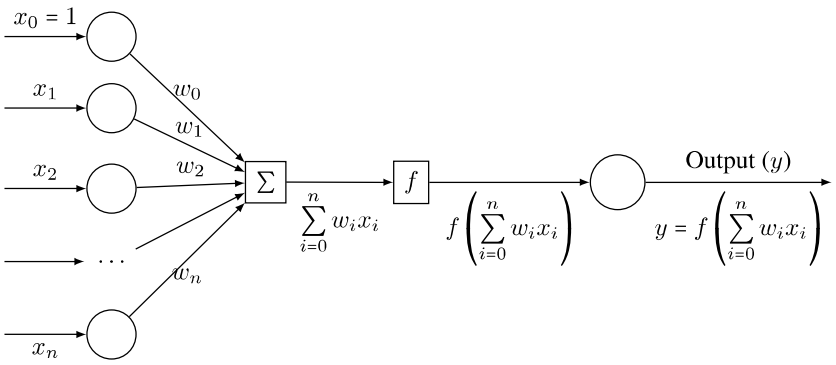


Figure 9.3: Schematic representation of an artificial neuron

$x_1, x_2, \ldots x_n$: input signals

$w_1, w_2, \ldots w_n$: weights associated with input signals

x_0 : input signal taking the constant value 1
w_0 : weight associated with x_0 (called bias)
$\sum$: indicates summation of input signals
f : function which produces the output
y : output signal

The function f can be expressed in the following form:

$$y = f\Big(\sum_{i=0}^{n} w_i x_i\Big) \tag{9.1}$$

Remarks

The small circles in the schematic representation of the artificial neuron shown in Figure 9.3 are called the *nodes* of the neuron. The circles on the left side which receives the values of $x_0, x_1, \ldots, x_n$ are called the *input nodes* and the circle on the right side which outputs the value of y is called *output node*. The squares represent the processes that are taking place before the result is outputted. They need not be explicitly shown in the schematic representation. Figure 9.4 shows a simplified representation of an artificial neuron.

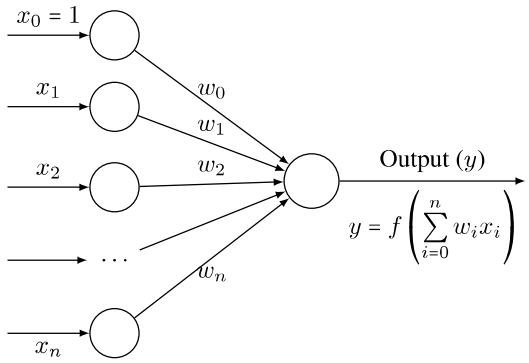


Figure 9.4: Simplified representation of an artificial neuron

9.4 Activation function

9.4.1 Definition

In an artificial neural network, the function which takes the incoming signals as input and produces the output signal is known as the *activation function*.

Remark

Eq.(9.1) represents the activation function of the ANN model shown in Figure **??**.

9.4.2 Some simple activation functions

The following are some of the simple activation functions.

1. Threshold activation function

The *threshold activation function* is defined by

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x \leq 0 \end{cases}$$

The graph of this function is shown in Figure 9.5.

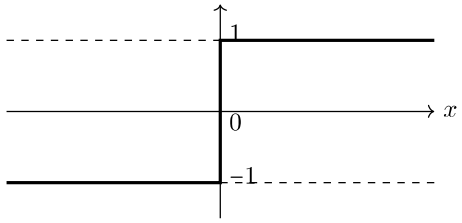


Figure 9.5: Threshold activation function

2. Unit step functions

Sometimes, the threshold activation function is also defined as a unit step function in which case it is called a *unit-step activation function*. This is defined as follows:

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

The graph of this function is shown in Figure 9.6.

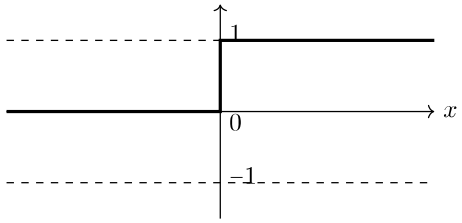


Figure 9.6: Unit step activation function

3. Sigmoid activation function (logistic function)

One of the must commonly used activation functions is the sigmoid activation function. It is defined as follows:

$$f(x) = \frac{1}{1 + e^{-x}}$$

The graph of the function is shown in Figure 9.7.

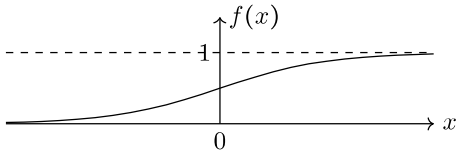


Figure 9.7: The sigmoid activation function

4. Linear activation function

The linear activation function is defined by

$$F(x) = mx + c.$$

This defines a straight line in the xy-plane.

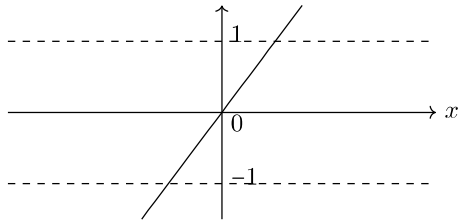


Figure 9.8: Linear activation function

5. Piecewise (or, saturated) linear activation function

This is defined by

$$f(x) = \begin{cases} 0 & \text{if } x < x_{\text{min}} \\ mx + c & \text{if } x_{\text{min}} \le x \le x_{\text{max}} \\ 0 & \text{if } x > x_{\text{max}} \end{cases}$$

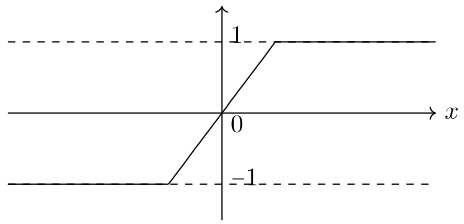


Figure 9.9: Piecewise linear activation function

6. Gaussian activation function

This is defined by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

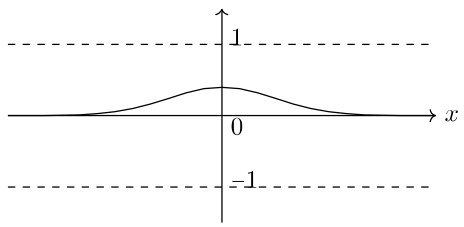


Figure 9.10: Gaussian activation function

7. Hyperbolic tangential activation function

This is defined by

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

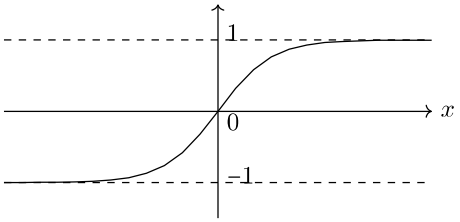


Figure 9.11: Hyperbolic tangent activation function

9.5 Perceptron

The perceptron is a special type of artificial neuron in which thee activation function has a special form.

9.5.1 Definition

A perceptron is an artificial neuron in which the activation function is the threshold function.

Consider an artificial neuron having $x_1, x_2, \cdots, x_n$ as the input signals and $w_1, w_2, \cdots, w_n$ as the associated weights. Let w_0 be some constant. The neuron is called a perceptron if the output of the neuron is given by the following function:

$$o(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + \cdots + w_nx_n > 0 \\ -1 & \text{if } w_0 + w_1x_1 + \cdots + w_nx_n \leq 0 \end{cases}$$

Figure 9.12 shows the schematic representation of a perceptron.

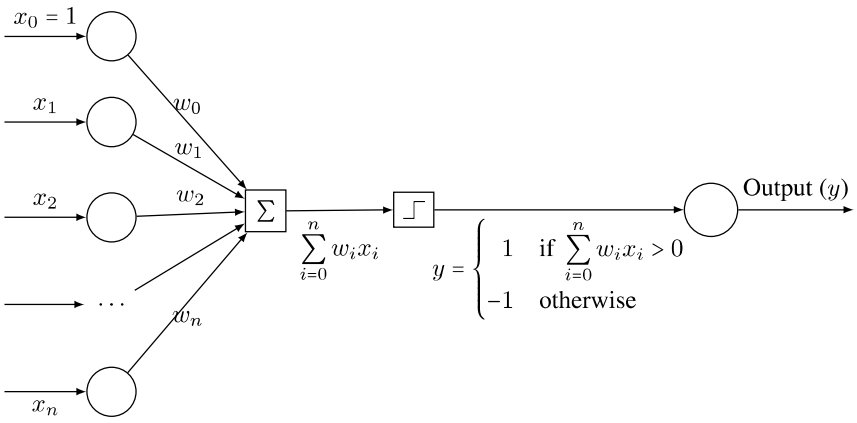


Figure 9.12: Schematic representation of a perceptrn

Remarks

1. The quantity $-w_0$ can be looked upon as a "threshold" that should be crossed by the weighted sum $w_1x_1 + \cdots + w_nx_n$ in order for the neuron to output a "1".

9.5.2 Representations of boolean functions by perceptrons

In this section we examine whether simple boolean functions like x_1 AND x_2 can be represented by perceptrons. To be consistent with the conventions in the definition of a perceptron we assume that the values -1 and 1 represent the boolean constants "false" and "true" respectively.

9.5.3 Representation of x_1 AND x_2

Let x_1 and x_2 be two boolean variables. Then the boolean function x_1 AND x_2 is represented by Table 9.1. It can be easily verified that the perceptron shown in Figure 9.13 represents the function

x_1	x_2	x_1 AND x_2
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

Table 9.1: The boolean function x_1 AND x_2

x_1 AND x_2.

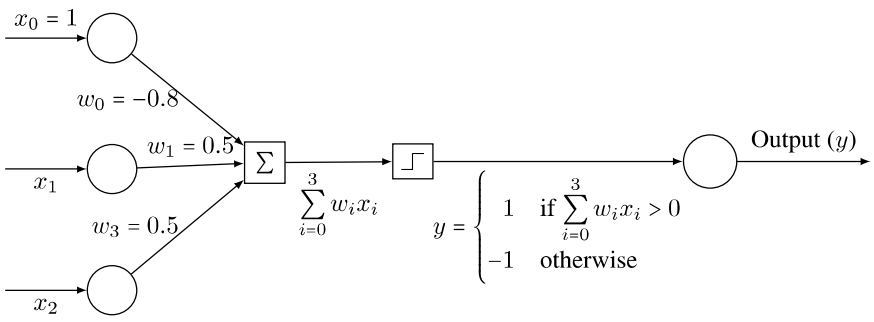


Figure 9.13: Representation of x_1 AND x_2 by a perceptron

In the perceptron shown in Figure 9.13, the output is given by

$$y = \begin{cases} 1 & \text{if } \sum_{i=0}^{3} w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$= \begin{cases} 1 & \text{if } -0.8 + 0.5x_1 + 0.5x_2 > 0 \\ -1 & \text{otherwise} \end{cases}$$

Representations of OR, NAND and NOR

The functions x_1 OR x_2, x_1 NAND x_2 and x_1 NOR x_2 can also be represented by perceptrons. Table 9.2 shows the values to be assigned to the weights w_0, w_1, w_2 for getting these boolean functions.

Boolean function	w_0	w_1	w_2
x_1 AND x_2	-0.8	0.5	0.5
x_1 OR x_2	-0.3	0.5	0.5
x_1 NAND x_2	0.8	-0.5	-0.5
x_1 NOR x_2	0.3	-0.5	-0.5

Table 9.2: Representations of boolean functions by perceptrons

Remarks

Not all boolean functions can be represented by perceptrons. For example, the boolean function x_1 XOR x_2 cannot be represented by a perceptron. This means that we cannot assign values to w_0, w_1, w_2 such that the expression $w_0 + w_1x_1 + w_2x_2$ takes the values of x_1 XOR x_2, and that this is the case can be easily verified also.

9.5.4 Learning a perceptron

By "learning a perceptron" we mean the process of assigning values to the weights and the threshold such that the perceptron produces correct output for each of the given training examples. The following are two algorithms to solve this learning problem:

9.5.5 Perceptron learning algorithm

Definitions

In the algorithm, we use the following notations:

n	:	Number of input variables
$y = f(\mathbf{z})$	:	Output from the perceptron for an input vector $\mathbf{z}$
$D = \{(\mathbf{x}_1, d_1), \ldots, (\mathbf{x}_s, d_s)\}$	:	Training set of s samples
$\mathbf{x}_j = (x_{j0}, x_{j1}, \ldots, x_{jn})$	:	The n-dimensional input vector
d_j	:	Desired output value of the perceptron for the input $\mathbf{x}_j$
x_{ji}	:	Value of the i-th feature of the j-th training input vector
x_{j0}	:	1
w_i	:	Weight of the i-th input variable
$w_i(t)$	:	Weight i at the t-th iteration

Algorithm

Step 1. Initialize the weights and the threshold. Weights may be initialized to 0 or to a small random value.

Step 2. For each example j in the training set D, perform the following steps over the input $\mathbf{x}_j$ and desired output d_j:

a) Calculate the actual output:

$$y_j(t) = f[w_0(t)x_{j0} + w_1(t)x_{j1} + w_2(t)x_{j2} + \cdots + w_n(t)x_{jn}]$$

b) Update the weights:

$$w_i(t+1) = w_i(t) + (d_j - y_j(t))x_{ji}$$

for all features $0 \leq i \leq n$.

Step 3. Step 2 is repeated until the iteration error $\frac{1}{s}\sum_{j=1}^{s}|d_j - y_j(t)|$ is less than a user-specified error threshold γ, or a predetermined number of iterations have been completed, where s is again the size of the sample set.

Remarks

The above algorithm can be applied only if the training examples are *linearly separable*.

9.6 Artificial neural networks

An *artificial neural network* (ANN) is a computing system inspired by the biological neural networks that constitute animal brains. An ANN is based on a collection of connected units called artificial neurons. Each connection between artificial neurons can transmit a signal from one to another. The artificial neuron that receives the signal can process it and then signal artificial neurons connected to it.

each connection between artificial neurons has a weight attached to it that get adjusted as learning proceeds. Artificial neurons may have a threshold such that only if the aggregate signal crosses that threshold the signal is sent. Artificial neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the input layer to the output layer, possibly after traversing the layers multiple times.

9.7 Characteristics of an ANN

An ANN can be defined and implemented in several different ways. The way the following characteristics are defined determines a particular variant of an ANN.

- **The activation function**

  This function defines how a neuron's combined input signals are transformed into a single output signal to be broadcasted further in the network.

- **The network topology (or architecture)**

  This describes the number of neurons in the model as well as the number of layers and manner in which they are connected.

- **The training algorithm**

  This algorithm specifies how connection weights are set in order to inhibit or excite neurons in proportion to the input signal.

9.7.1 Activation functions

The activation function is the mechanism by which the artificial neuron processes incoming information and passes it throughout the network. Just as the artificial neuron is modeled after the biological version, so is the activation function modeled after nature's design.

Let $x_1, x_2, \ldots, x_n$ be the input signals, $w_1, w_2, \ldots, w_n$ be the associated weights and $-w_0$ the threshold. Let

$$x = w_0 + w_1x_1 + \cdots + w_nx_n.$$

The activation function is some function of x. Some of the simplest and commonly used activations are given in Section 9.4.

9.7.2 Network topology

By "network topology" we mean the patterns and structures in the collection of interconnected nodes. The topology determines the complexity of tasks that can be learned by the network. Generally, larger and more complex networks are capable of identifying more subtle patterns and complex decision boundaries. However, the power of a network is not only a function of the network size, but also the way units are arranged.

Different forms of forms of network architecture can be differentiated by the following characteristics:

- The number of layers
- Whether information in the network is allowed to travel backward
- The number of nodes within each layer of the network

1. The number of layers

In an ANN, the *input nodes* are those nodes which receive unprocessed signals directly from the input data. The *output nodes* (there may be more than one) are those nodes which generate the final predicted values. A *hidden node* is a node that processes the signals from the input nodes (or other such nodes) prior to reaching the output nodes.

The nodes are arranged in *layers*. The set of nodes which receive the unprocessed signals from the input data constitute the *first layer* of nodes. The set of hidden nodes which receive the outputs from the nodes in the first layer of nodes constitute the *second layer* of nodes. In a similar way we can define the third, fourth, etc. layers. Figure 9.14 shows an ANN with only one layer of nodes. Figure 9.15 shows an ANN with two layers.

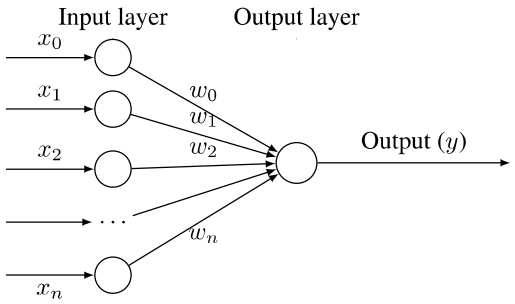


Figure 9.14: An ANN with only one layer

2. The direction of information travel

Networks in which the input signal is fed continuously in one direction from connection to connection until it reaches the output layer are called *feedforward networks*. The network shown in Figure 9.15 is a feedforward network.

Networks which allows signals to travel in both directions using loops are called *recurrent networks* (or, *feedback networks*).

In spite of their potential, recurrent networks are still largely theoretical and are rarely used in practice. On the other hand, feedforward networks have been extensively applied to real-world problems. In fact, the multilayer feedforward network, sometimes called the Multilayer Perceptron (MLP), is the de facto standard ANN topology. If someone mentions that they are fitting a neural network, they are most likely referring to a MLP.

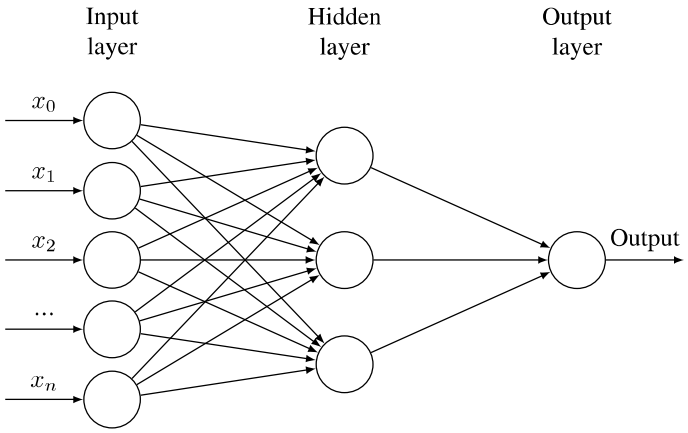


Figure 9.15: An ANN with two layers

3. The number of nodes in each layer

The number of input nodes is predetermined by the number of features in the input data. Similarly, the number of output nodes is predetermined by the number of outcomes to be modeled or the number of class levels in the outcome. However, the number of hidden nodes is left to the user to decide prior to training the model. Unfortunately, there is no reliable rule to determine the number of neurons in the hidden layer. The appropriate number depends on the number of input nodes, the amount of training data, the amount of noisy data, and the complexity of the learning task, among many other factors.

9.7.3 The training algorithm

There are two commonly used algorithms for learning a single perceptron, namely, the perceptron rule and the delta rule. The former is used when the training data set is linearly separable and the latter when the training data set is not linearly separable.

The algorithm which is now commonly used to train an ANN is known simply as *backpropagation*.

9.7.4 The cost function

Definition

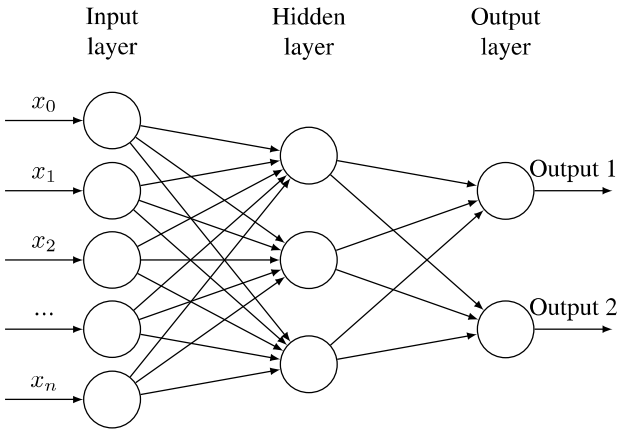
In a machine learning algorithm, the *cost function* is a function that measures how well the algorithm maps the target function that it is trying to guess or a function that determines how well the algorithm performs in an optimization problem.

Remaarks

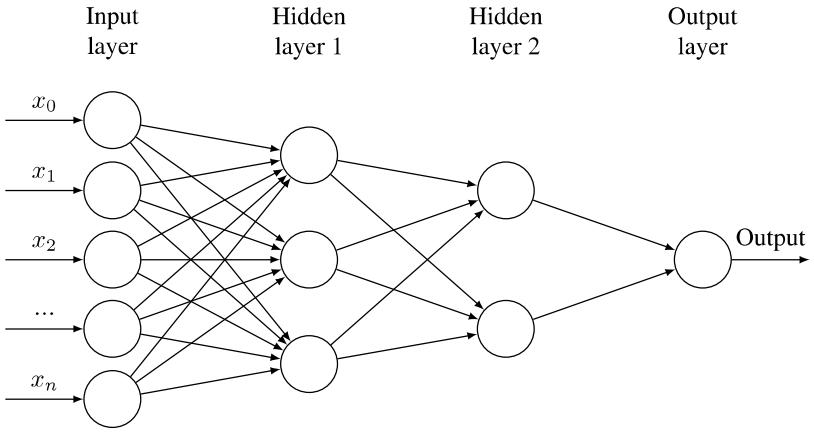
The cost function is also called the *loss function*, the *objective function*, the *scoring function*, or the *error function*.

Example

Let y be the the output variable. Let $y_1, \ldots, y_n$ be the actual values of y in n examples and $\hat{y}_1, \ldots, \hat{y}_n$ be the values predicted by an algorithm.



(a) Network with one hidden layer and two output nodes



(b) Network with two hidden layers

Figure 9.16: Examples of different topologies of networks

1. The sum of squares of the differences between the predicted and actual values of y, denoted by SSE and defined below, can be taken as a cost function for the algorithm.

$$\text{SSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

2. The mean of the sum of squares of the differences between the predicted and actual values of y, denoted by MSE and defined below, can be taken as a cost function for the algorithm.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

9.8 Backpropagation

The backpropagation algorithm was discovered in 1985-86. Here is an outline of the algorithm.

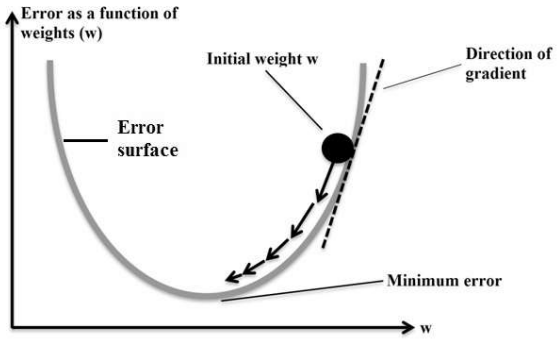


Figure 9.17: A simplified model of the error surface showing the direction of gradient

9.8.1 Outline of the algorithm

1. Initially the weights are assigned at random.
2. Then the algorithm iterates through many cycles of two processes until a stopping criterion is reached. Each cycle is known as an *epoch*. Each epoch includes:
   (a) A *forward phase* in which the neurons are activated in sequence from the input layer to the output layer, applying each neuron's weights and activation function along the way. Upon reaching the final layer, an output signal is produced.
   (b) A *backward phase* in which the network's output signal resulting from the forward phase is compared to the true target value in the training data. The difference between the network's output signal and the true value results in an error that is propagated backwards in the network to modify the connection weights between neurons and reduce future errors.
3. The technique used to determine how much a weight should be changed is known as *gradient descent method*. At every stage of the computation, the error is a function of the weights. If we plot the error against the wights, we get a higher dimensional analog of something like a curve or surface. At any point on this surface, the gradient suggests how steeply the error will be reduced or increased for a change in the weight. The algorithm will attempt to change the weights that result in the greatest reduction in error (see Figure 9.17).

9.8.2 Illustrative example

To illustrate the various steps in the backpropagation algorithm, we consider a small network with two inputs, two outputs and one hidden layer as shown in Figure 9.18.[2]

We assume that there are two observations:

Sample	Input 1 i_1	Input 2 i_2	Output target 1 T_1	Output target 2 T_2
1	0.05	0.10	0.01	0.99
2	0.25	0.18	0.23	0.79

We are required to estimate the optimal values of the weights $w_1, \ldots, w_8, b_1, b_2$. Here b_1 and b_2 are the biases. For simplicity, we have assigned the same biases to both nodes in the same layer.

Step 1. We initialise the connection weights to small random values. These initial weights are shown in Figure 9.19.

[2]Thanks to `https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/` for this example.

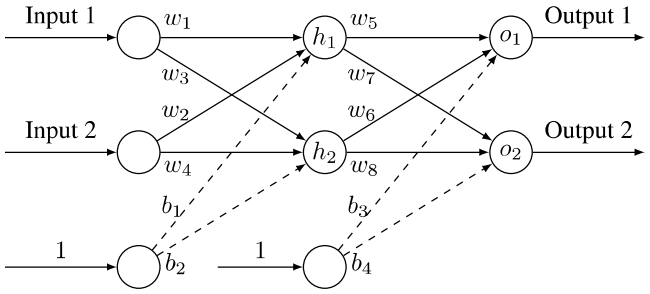


Figure 9.18: ANN for illustrating backpropagation algorithm

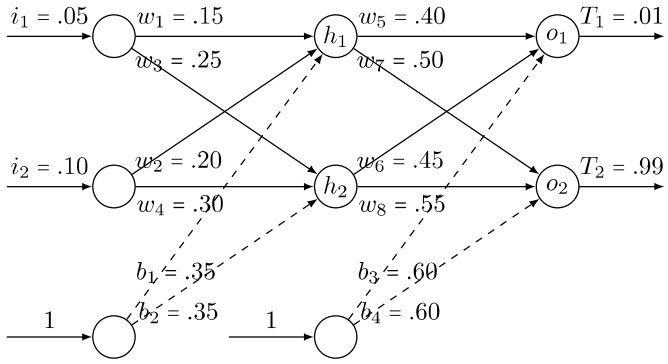


Figure 9.19: ANN for illustrating backpropagation algorithm with initial values for weights

Step 2. Present the first sample inputs and the corresponding output targets to the network. This is shown in Figure 9.19.

Step 3. Pass the input values to the first layer (the layer with nodes h_1 and h_2).

Step 4. We calculate the outputs from h_1 and h_2. We use the logistic activation function

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

$$\begin{aligned}
\text{out}_{h_1} &= f(w_1 \times i_1 + w_2 \times i_2 + b_1 \times 1) \\
&= f(0.15 \times 0.05 + 0.20 \times 0.10 + 0.35 \times 1) \\
&= f(0.3775) \\
&= \frac{1}{1+e^{-0.3775}} \\
&= 0.59327 \\
\text{out}_{h_2} &= f(w_3 \times i_1 + w_4 \times i_2 + b_2 \times 1) \\
&= f(0.25 \times 0.05 + 0.30 \times 0.10 + 0.35 \times 1) \\
&= f(0.3925) \\
&= \frac{1}{1+e^{-0.3925}} \\
&= 0.59689
\end{aligned}$$

Step 5. We repeat this process for every layer. We get the outputs from the nodes in the output layer as follows:

$$\begin{aligned}
\text{out}_{o_1} &= f(w_5 \times \text{out}_{h_1} + w_6 \times \text{out}_{h_2} + b_3 \times 1) \\
&= f(0.40 \times 0.59327 + 0.45 \times 0.59689 + 0.60 \times 1) \\
&= f(1.10591) \\
&= \frac{1}{1 + e^{-1.10591}} \\
&= 0.75137 \\
\text{out}_{o_2} &= f(w_7 \times \text{out}_{h_1} + w_8 \times \text{out}_{h_2} + b_4 \times 1) \\
&= f(0.50 \times 0.59327 + 0.55 \times 0.59689 + 0.60 \times 1) \\
&= f(1.22492) \\
&= \frac{1}{1 + e^{-1.22492}} \\
&= 0.77293
\end{aligned}$$

The sum of the squares of the output errors is given by

$$\begin{aligned}
E &= \frac{1}{2}(T_1 - \text{out}_{o_1})^2 + \frac{1}{2}(T_2 - \text{out}_{o_2})^2 \\
&= (0.01 - 0.75137)^2 + (0.99 - 0.77293)^2 \\
&= 0.298371
\end{aligned}$$

Step 6. We begin backward phase. We adjust the weights. We first adjust the weights leading to the nodes o_1 and o_2 in the output layer and then the weights leading to the nodes h_1 and h_2 in the hidden layer. The adjusted values of the weights $w_1, \ldots, w_8, b_1, \ldots, b_4$ are denoted by $w_1^+, \ldots, w_8^+, b_1^+, \ldots, b_4^+$. The computations use a certain constant η called the *learning rate*. In the following we have taken $\eta = 0.5$.

(a) Computation of adjusted weights leading to o_1 and o_2:

$$\begin{aligned}
\delta_{o_1} &= (T_1 - \text{out}_{o_1}) \times \text{out}_{o_1} \times (1 - \text{out}_{o_1}) \\
&= (0.01 - 0.75137) \times 0.75137 \times (1 - 0.75137) \\
&= -0.13850 \\
w_5^+ &= w_5 + \eta \times \delta_{o_1} \times \text{out}_{h_1} \\
&= 0.40 + 0.5 \times (-0.13850) \times 0.59327 \\
&= 0.35892 \\
w_6^+ &= w_6 + \eta \times \delta_{o_1} \times \text{out}_{h_2} \\
&= 0.45 + 0.5 \times (-0.13850) \times 0.59689 \\
&= 0.40867 \\
b_3^+ &= b_3 + \eta \times \delta_{o_1} \times 1 \\
&= 0.60 + 0.5 \times (-0.13850) \times 1 \\
&= 0.53075
\end{aligned}$$

$$\begin{aligned}
\delta_{o_2} &= (T_2 - \text{out}_{o_2}) \times \text{out}_{o_2} \times (1 - \text{out}_{o_2}) \\
&= (0.99 - 0.77293) \times 0.77293 \times (1 - 0.77293) \\
&= 0.03810 \\
w_7^+ &= w_7 + \eta \times \delta_{o_2} \times \text{out}_{h_1} \\
&= 0.50 + 0.5 \times 0.03810 \times 0.59327
\end{aligned}$$

$$
\begin{aligned}
&= 0.51130 \\
w_8^+ &= w_8 + \eta \times \delta_{o_2} \times \text{out}_{h_2} \\
&= 0.55 + 0.5 \times 0.03810 \times 0.59689 \\
&= 0.56137 \\
b_4^+ &= b_4 + \eta \times \delta_{o_2} \times 1 \\
&= 0.60 + 0.5 \times 0.03810 \times 1 \\
&= 0.61905
\end{aligned}
$$

(b) Computation of adjusted weights leading to h_1 and h_2:

$$
\begin{aligned}
\delta_{h_1} &= (\delta_{o_1} \times w_5 + \delta_{o_2} \times w_7) \times \text{out}_{h_1} \times (1 - \text{out}_{h_1}) \\
&= (-0.13850 \times 0.40 + 0.03810 \times 0.50) \times 0.59327 \times (1 - 0.59327) \\
&= -0.00877 \\
w_1^+ &= w_1 + \eta \times \delta_{h_1} \times i_1 \\
&= 0.15 + 0.5 \times (-0.00877) \times 0.05 \\
&= 0.14978 \\
w_2^+ &= w_2 + \eta \times \delta_{h_1} \times i_2 \\
&= 0.20 + 0.5 \times (-0.00877) \times 0.10 \\
&= 0.19956 \\
b_1^+ &= b_1 + \eta \times \delta_{h_1} \times 1 \\
&= 0.35 + 0.5 \times (-0.00877) \times 1 \\
&= 0.34562
\end{aligned}
$$

$$
\begin{aligned}
\delta_{h_2} &= (\delta_{o_1} \times w_6 + \delta_{o_2} \times w_8) \times \text{out}_{h_2} \times (1 - \text{out}_{h_2}) \\
&= ((-0.13850) \times 0.45 + 0.03810 \times 0.55) \times 0.59689 \times (1 - 0.59689) \\
&= -0.00995 \\
w_3^+ &= w_3 + \eta \times \delta_{h_2} \times i_1 \\
&= 0.25 + 0.5 \times (-0.00995) \times 0.05 \\
&= 0.24975 \\
w_4^+ &= w_4 + \eta \times \delta_{h_2} \times i_2 \\
&= 0.30 + 0.5 \times (-0.00995) \times 0.10 \\
&= 0.29950 \\
b_2^+ &= b_2 + \eta \times \delta_{h_2} \times 1 \\
&= 0.35 + 0.5 \times (-0.00995) \times 1 \\
&= 0.34503
\end{aligned}
$$

Step 7. Now we set:

$$
\begin{gathered}
w_1 = w_1^+, \quad w_2 = w_2^+, \quad w_3 = w_3^+, \quad w_4 = w_4^+ \\
w_5 = w_5^+, \quad w_6 = w_6^+, \quad w_7 = w_7^+, \quad w_8 = w_8^+ \\
b_1 = b_1^+, \quad b_2 = b_2^+, \quad b_3 = b_3^+, \quad b_4 = b_4^+
\end{gathered}
$$

We choose the next sample input and the corresponding output targets to the network and repeat Steps 2 to 6.

Step 8. The process in Step 7 is repeated until the root mean square of output errors is minimised.

Remarks

1. The constant $\frac{1}{2}$ is included in the expression for E so that the exponent is cancelled when we differentiate it. The result has been multiplied by a learning rate $\eta = 0.5$ and so it doesnâĂŹt matter that we introduce the constant $\frac{1}{2}$ in E.

2. In the above computations, the method used to calculate the adjusted weights is known as the *delta rule*.

3. The rule for computing the adjusted weights can be succinctly stated as follows. Let w be a weight and w^+ its adjusted weight. Let E be the the total sum of squares of errors. Then w^+ is computed by
$$w^+ = w - \eta \frac{\partial E}{\partial w}.$$
Here $\frac{\partial E}{\partial w}$ is the gradient of E with respect to w; that is, the rate at which E is changing with respect to w. (The set of all such gradients specifies the direction in which E is decreasing the most rapidly, that is, the direction of quickest descent.) For example, it can be shown that
$$\begin{aligned}\frac{\partial E}{\partial w_5} &= -(T_1 - \text{out}_{o_1}) \times \text{out}_{o_1} \times (1 - \text{out}_{o_1}) \times \text{out}_{h_1} \\ &= -\delta_{o_1} \times \text{out}_{h_1}\end{aligned}$$
and so
$$\begin{aligned}w_5^+ &= w_5 - \eta \frac{\partial E}{\partial w_5} \\ &= w_5 + \eta \times \delta_{o_1} \times \text{out}_{h_1}\end{aligned}$$

9.8.3 The algorithm

The backpropagation algorithm trains a given feed-forward multilayer neural network for a given set of input patterns with known classifications. When each entry of the sample set is presented to the network, the network examines its output response to the sample input pattern. The output response is then compared to the known and desired output and the error value is calculated. Based on the error, the connection weights are adjusted. The adjustments are based on the mean square error of the output response to the sample input and it is known as the *delta learning rule*. The set of these sample patterns are repeatedly presented to the network until the error value is minimized.

Notations

Figures 9.20 and 9.21 show the various notations used in the algorithm.

M	: Number of layers (excluding the input layer which is assigned the layer number 0)
N_j	: Number of neurons (nodes) in j-th layer
$\mathbf{X}_p = (X_{p1}, X_{p2}, \ldots, X_{pN_0})$	: p-th training sample
$\mathbf{T}_p = (T_{p1}, T_{p2}, \ldots, T_{pN_M})$	: Known output corresponding to the p-th training sample
$\mathbf{O}_p = (O_{p1}, O_{p2}, \ldots, O_{pN_M})$	: Actual output by the network corresponding to the p-th training sample
Y_{ji}	: Output from the i-th neuron in layer j
W_{jik}	: Connection weight from k-th neuron in layer $(j-1)$ to i-th neuron in layer j
δ_{ji}	: Error value associated with the i-th neuron in layer j

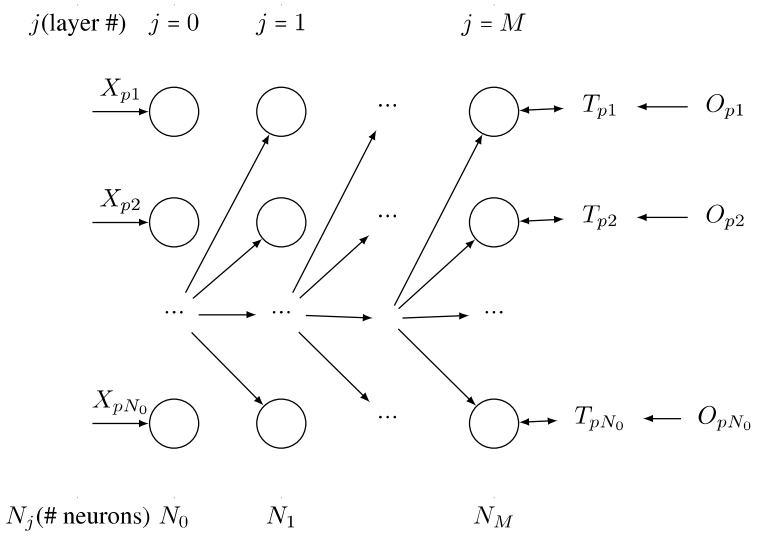


Figure 9.20: Notations of backpropagation algorithm

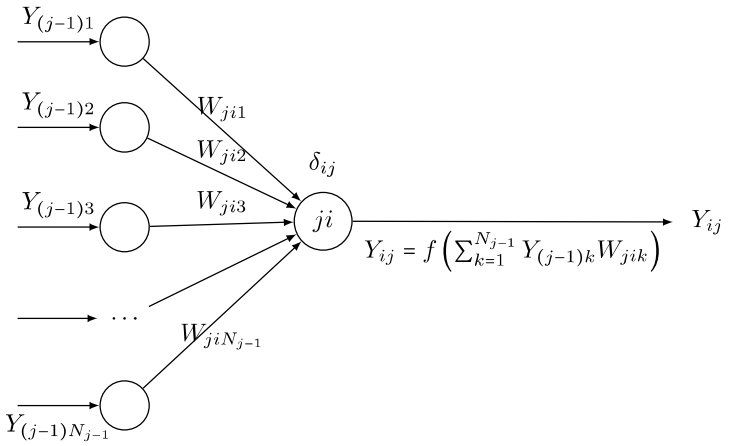


Figure 9.21: Notations of backpropagation algorithm: The i-th node in layer j

The algorithm

Step 1. Initialize connection weights into small random values.

Step 2. Present the pth sample input vector of pattern

$$\mathbf{X}_p = (X_{p1}, X_{p2}, ..., X_{pN_0})$$

and the corresponding output target

$$\mathbf{T}_p = (T_{p1}, T_{p2}, ..., T_{pN_M})$$

to the network.

Step 3. Pass the input values to the first layer, layer 1. For every input node i in layer 0, perform: $Y_{0i} = X_{pi}$.

Step 4. For every neuron i in every layer $j = 1, 2, ..., M$, find the output from the neuron:

$$Y_{ji} = f\left(\sum_{k=1}^{N_{j-1}} Y_{(j-1)k} W_{jik}\right),$$

where

$$f(x) = \frac{1}{1 + \exp(-x)}.$$

Step 5. Obtain output values. For every output node i in layer M, perform:

$$O_{pi} = Y_{Mi}.$$

Step 6. Calculate error value δ_{ji} for every neuron i in every layer in backward order $j = M, M - 1, \ldots, 2, 1$, from output to input layer, followed by weight adjustments. For the output layer, the error value is:

$$\delta_{Mi} = Y_{Mi}(1 - Y_{Mi})(T_{pi} - Y_{Mi}),$$

and for hidden layers:

$$\delta_{ji} = Y_{ji}(1 - Y_{ji}) \sum_{k=1}^{N_{j+1}} \delta_{(j+1)k} W_{(j+1)ki}.$$

The weight adjustment can be done for every connection from neuron k in layer $(j - 1)$ to every neuron j in every layer i:

$$W_{jik}^{+} = W_{jik} + \eta \delta_{ji} Y_{ji},$$

where η represents weight adjustment factor (called the *learning rate*) normalized between 0 and 1.

Step 7. The actions in steps 2 through 6 will be repeated for every training sample pattern p, and repeated for these sets until the sum of the squares of output errors is minimized.

9.9 Introduction to deep learning

9.9.1 Definition

A neural network with multiple hidden layers is called a *Deep Neural Network* (DNN) and the practice of training such network is referred to as *deep learning*.

Remarks

In the terminology "deep learning", the term "deep" is a technical term. It refers to the number of layers in a neural network. A *shallow network* has one so-called hidden layer, and a deep network has more than one. Multiple hidden layers allow deep neural networks to learn features of the data in a so-called feature hierarchy, because simple features recombine from one layer to the next, to form more complex features. Networks with many layers pass input data (features) through more mathematical operations than networks with few layers, and are therefore more computationally intensive to train. Computational intensivity is one of the hallmarks of deep learning.

Figure 9.22 shows a shallow neural network and Figure 9.23 shows a deep neural network with three hidden layers.