Chapter 11
Neural Networks

As we have mentioned in the previous chapter, the neural network model (NN) is sometimes treated as one of the three approaches to pattern recognition (along with the approach introduced in the previous chapter and syntactic-structural pattern recognition). In fact, as we will see in this chapter, various models of (artificial) neural networks are analogous to standard pattern recognition methods, in the sense of their mathematical formalization.[1]

Nevertheless, in spite of these analogies, neural network theory is distinguished from standard pattern recognition because of the former original methodological foundations (connectionism), the possibility of implementing standard algorithms with the help of network architectures and a variety of learning techniques.

A lot of different models of neural networks have been developed till now. A taxonomy of these models is usually troublesome for beginners in the area of neural networks. Therefore, in this chapter we introduce notions in a hierarchical ("bottom-up") step-by-step way. In the first section we introduce a generic model of a neuron and we consider the criteria used for defining a typology of artificial neurons. Basic types of neural networks are discussed in Sect. 2. A short survey of the most popular specific models of neural networks is presented in the last section.

[1] Anil K. Jain—an eminent researcher in both these areas of Artificial Intelligence pointed out these analogies in a paper [148] published in 2000. Thus, there are the following analogies: linear discriminant functions—one-layer perceptron, Principal Component Analysis—auto-associative NN, non-linear discriminant functions—multilayer perceptron, etc.

M. Flasiński, *Introduction to Artificial Intelligence*,
DOI 10.1007/978-3-319-40022-8_11

11.1 Artificial Neuron

At the end of the nineteenth century Santiago Ramón y Cajal[2] discovered that a brain consists of neural cells, then called *neurons*. The structure of a neuron is shown in Fig. 11.1a. A neuron consists of a *cell body* (called the *perikaryon* or *soma*) and cellular extensions of two types. Extensions called *dendrites* are thin branching structures. They are used for the transmission of signals from other neurons to the cell body. An extension called an *axon* transmits a signal from the cell body to other neurons.

Communication among neurons is done by transmitting electrical or chemical signals with the help of *synapses*. Research led by John Carew-Eccles[3] discovered the mechanism of communication among neurons. Transmission properties of synapses are *controlled* by chemicals called *neurotransmitters* and synaptic signals can be excitatory or inhibitory.

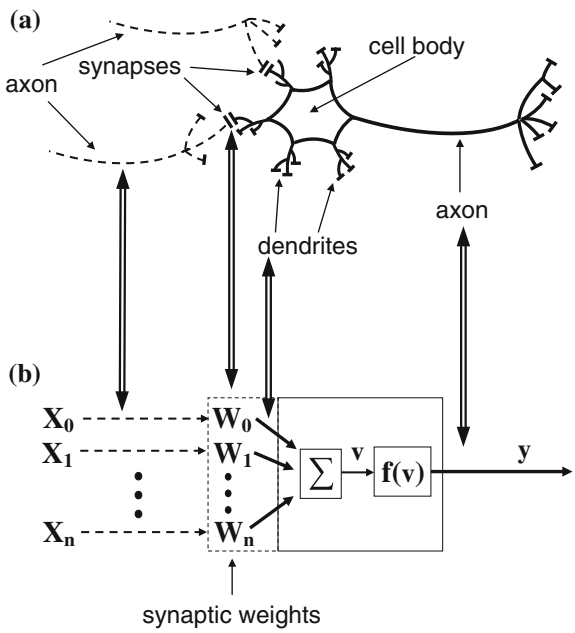

Fig. 11.1 **a** The structure of a neuron, **b** the scheme of an artificial neuron

[2]Santiago Ramón y Cajal—an eminent histologist and neuroscientist, a professor of universities in Valenzia, Barcelona, and Madrid. In 1906 he received the Nobel Prize (together with Camillo Golgi) for research into neural structures.

[3]John Carew Eccles—a professor of neurophysiology at the University of Otago (New Zealand), Australian National University, and the University at Buffalo. In 1963 he was awarded the Nobel Prize for research into synaptic transmission.

At the same time, Alan Lloyd Hodgkin[4] and Andrew Fielding Huxley[5] led research into the process of initiating an action potential, which plays a key role in communication among neurons. They performed experiments on the huge axon of an Atlantic squid. The experiments allowed them to discover the mechanism of this process. At the moment when the total sum[6] of excitatory postsynaptic potential reaches a certain threshold, an *action potential* occurs in the neuron. The action potential is then emitted by the neuron (we say that the neuron *fires*) and it is propagated via its axon to other neurons. As was later shown by Bernard Katz[7] the action potential is generated according to the *all or none* principle, i.e., either it occurs fully or it does not occur at all. In Fig. 11.1a a neuron as described above is marked with a solid line, whereas axons belonging to two other neurons which send signals to it are marked with dashed lines.

As we have mentioned already in Sect. 3.1, an (artificial) *neural network, NN*, is a (simplified) model of a brain treated as a structure consisting of neurons. The model of an *artificial neuron* was developed by Warren S. McCulloch and Walter Pitts in 1943 [198]. It is shown in Fig. 11.1b. We describe its structure and behavior on the basis of the notions which have been introduced for a biological neuron above. Input signals $X_0, X_1, \ldots, X_n$ correspond to neural signals sent from other neurons. We assume (for technical reasons) that $X_0 = 1$. These signals are represented by an *input vector* $\mathbf{X} = (X_0, X_1, \ldots, X_n)$.

In order to compute the total sum of affecting input signals on the neuron, we introduce a *postsynaptic potential function* g. In our considerations we assume that the function g is in the form of a sum. This means that input signals are multiplied by *synaptic weights* $W_0, W_1, \ldots, W_n$, which define a *weight vector* $\mathbf{W} = (W_0, W_1, \ldots, W_n)$. Synaptic weights play the role of the *controller* of transmission properties of synapses by analogy to a biological neuron. The weights set some inputs to be *excitatory synapses* and some to be *inhibitory synapses*. The multiplication of input signals by weights corresponds to the enhancement or weakening of signals sent to the neuron from other neurons. After the multiplication of input signals by weights, we sum the products, which gives a signal v:

$$v = g(\mathbf{W}, \mathbf{X}) = \sum_{i=0}^{n} W_i X_i. \tag{11.1}$$

[4] Alan Lloyd Hodgkin—a professor of physiology and biophysics at the University of Cambridge. In 1963 he was awarded the Nobel Prize for research into nerve action potential. He was the President of the Royal Society.

[5] Andrew Fielding Huxley—a professor of physiology and biophysics at the University of Cambridge. In 1963 he was awarded the Nobel Prize (together with Alan Lloyd Hodgkin). He was a grandson of the biologist Thomas H. Huxley, who was called "Darwin's Bulldog" because he vigorously supported the theory of evolution during a famous debate with the Bishop of Oxford Samuel Wilberforce in 1860.

[6] In the sense that multiple excitatory synapses act on the neuron.

[7] Bernard Katz—a professor of biophysics at University College London. He was awarded the Nobel Prize in physiology and medicine in 1970.

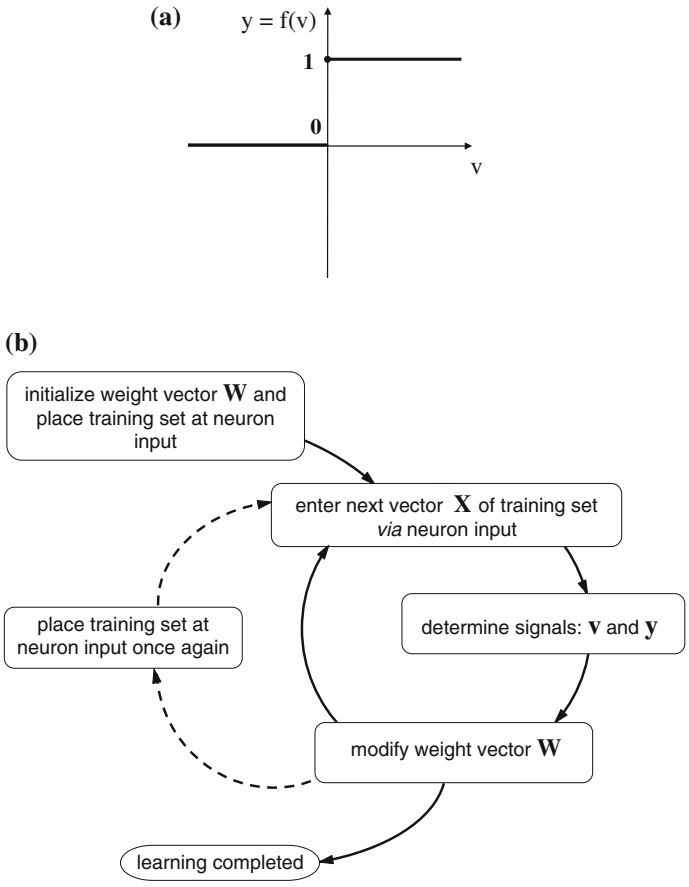

Fig. 11.2 **a** The activation function of the McCulloch-Pitts neuron, **b** the general scheme of neuron learning

This signal corresponds to the total sum of excitatory postsynaptic potential. Thus, we have to check whether it has reached the proper threshold which is required for activating the neuron. We do this with the help of the *activation (transfer) function* f, which generates an *output signal* y for a signal v, i.e.,

$$y = f(v). \tag{11.2}$$

McCulloch and Pitts used the *Heaviside step function*, usually denoted by $\mathbf{1}(v)$ (cf. Fig. 11.2a), as the activation function. It is defined in the following way:

$$\mathbf{1}(v) = \begin{cases} 1, & \text{if } v \geq 0, \\ 0, & \text{if } v < 0. \end{cases} \tag{11.3}$$

As we can see the Heaviside step function gives 0 for values of a signal v which are less than zero, otherwise it gives 1. Thus, the threshold is set to 0. In fact, we can set any threshold with the help of the synaptic weight W_0, since we have assumed that the signal $X_0 = 1$.

In the generic neuron model the *output function out* is the third function (in a sequence of signal processing). It is used in advanced models. In the monograph we assume that it is the identity function, i.e., $out(y) = y$. Therefore, we omit it in further considerations.

The brain is an organ which can learn, so (artificial) neural networks also should have this property. The general scheme of *neuron learning* is shown in Fig. 11.2b. A neuron should learn to react in a proper way to *patterns* (i.e., feature vectors of patterns) that are shown to it.[8] We begin with a random initialization of the weight vector $\mathbf{W}$ of the neuron. We place the training set at the neuron input. Then, we start the main cycle of the learning process.

We enter the next vector $\mathbf{X}$ of the training set[9] via the neuron input. The neuron computes a value v for this vector according to a formula (11.1) and then it determines the value y according to the given activation function.

The output signal y is the reaction of the neuron to a pattern which has been shown. The main idea of the learning process consists of modifying the weights of the neuron, depending on its reaction to the pattern shown. This is done according to the chosen learning method.[10] Thus, in the last step of the main cycle we modify the weight vector $\mathbf{W}$ of the neuron. Then, we enter the next pattern of the training set, etc.

After showing all feature vectors of the training set to the neuron, we can decide whether it has learned to recognize patterns. We claim it has learned to recognize patterns if its weights are set in such a way that it reacts to patterns in a correct way. If not, we have to repeat the whole cycle of the learning process, i.e., we have to place the training set at the neuron input once again and we have to begin showing vectors once again. In Fig. 11.2b this is marked with dashed arrows.

Methods of neuron learning can be divided into the following two groups:

- supervised learning,
- unsupervised learning.

In *supervised learning* the training set is of the form:

$$U = (\ (\mathbf{X}(1), u(1)), (\mathbf{X}(2), u(2)), \ldots, (\mathbf{X}(M), u(M))\ ), \tag{11.4}$$

where $\mathbf{X}(j) = (X_0(j), X_1(j), \ldots, X_n(j))$, $j = 1, \ldots, M$, is the jth input vector and $u(j)$ is the signal which should be generated by the neuron after input of this vector (according to the opinion of a *teacher*). We say that the neuron reacts properly

[8] Showing patterns means entering their feature vectors *via* the neuron input.

[9] The first one is the first vector of the training set.

[10] Basic learning methods are introduced later.

to the vectors shown, if for each pattern $\mathbf{X}(j)$ it generates an output signal $y(j)$ which is equal to the signal $u(j)$ required by the teacher (accurate within a small error).

In *unsupervised learning* the training set is of the form:

$$U = (\,\mathbf{X}(1), \mathbf{X}(2), \ldots, \mathbf{X}(M)\,). \tag{11.5}$$

In this case the neuron should modify the weights itself in such a way that it generates the same output signal for *similar* patterns and it generates various output signals for patterns which are *different* from one another.[11]

Let us consider supervised learning with the example of the *perceptron* introduced by Frank Rosenblatt in 1957 [246]. The *bipolar step function* is used as the activation function for the perceptron. It is defined in the following way (cf. Fig. 11.3):

$$f(v) = \begin{cases} 1, & \text{if } v > 0, \\ -1, & \text{if } v \leq 0. \end{cases} \tag{11.6}$$

where v is computed according to formula (11.1). Learning, i.e., modifying the perceptron weights, is performed according to the following principle.
If at the jth step of learning $y(j) \neq u(j)$, then new weights (for the $(j+1)$th step) are computed according to the following formula:

$$W_i(j+1) = W_i(j) + u(j)X_i(j), \tag{11.7}$$

where $X_i(j)$ is the ith coordinate of the vector shown in the jth step and $u(j)$ is the output signal required for this vector. Otherwise, i.e., if $y(j) = u(j)$, the weights do not change, i.e., $W_i(j+1) = W_i(j)$.[12]

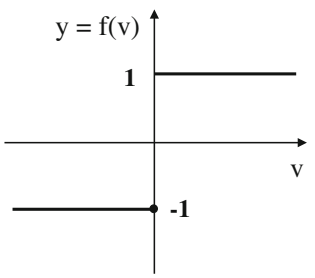

Fig. 11.3 The activation function of the perceptron

[11] The reader will easily notice analogies to *pattern recognition* and *cluster analysis* which have been introduced in the previous chapter.

[12] In order to avoid confusing the indices of the training set elements we assume that after starting a new cycle of learning we re-index these elements, i.e., they take the subsequent indices. Of course, the first weight vector of the new cycle is computed on the basis of the last weight vector of the previous one.

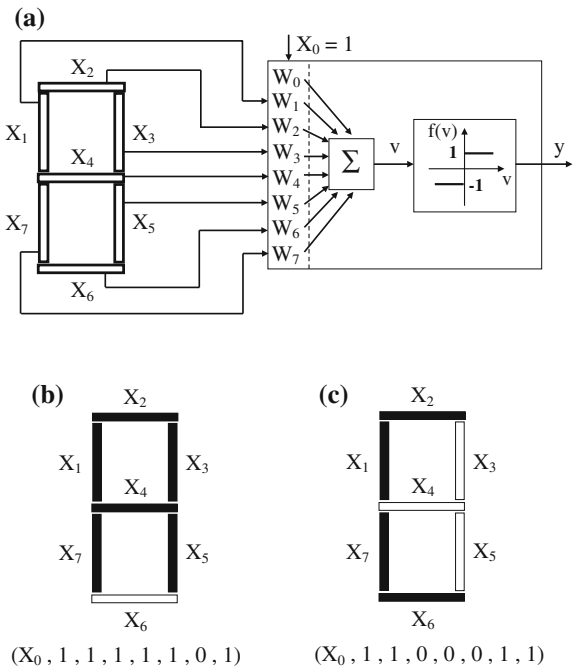

Fig. 11.4 **a** The connection of LED display segments to the perceptron input, **b** the display of the character *A* and the corresponding feature vector, **c** the display of the character *C* and the corresponding feature vector

Let us assume that we would like to use a perceptron for recognizing characters which are shown by an LED display as depicted in Fig. 11.4a. The display consists of seven segments. If a segment is switched on, then it sends a signal to the perceptron according to the scheme of connections which is shown in Fig. 11.4a. Thus, we can denote the input vector by $\mathbf{X} = (X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7)$.

Let us assume that we would like to teach the perceptron to recognize two characters, *A* and *C*.[13] These characters are represented by the following input signals: $\mathbf{X}_A = (X_0, 1, 1, 1, 1, 1, 0, 1)$ and $\mathbf{X}_C = (X_0, 1, 1, 0, 0, 0, 1, 1)$[14] (cf. Fig. 11.4b, c). In the case of the character *A* the perceptron should generate an output signal $u = 1$ and in the case of the character *C* the perceptron should generate an output signal $u = -1$. Let us assume that $\mathbf{W}(1) = (W_0(1), W_1(1), W_2(1), W_3(1), W_4(1), W_5(1), W_6(1), W_7(1)) = (0, 0, 0, 0, 0, 0, 0, 0)$ is the initial weight vector.[15]

Let us track the subsequent steps of the learning process.

Step 1. The character *A*, i.e., the feature vector (1, 1, 1, 1, 1, 1, 0, 1) is shown to the perceptron. We compute a value v on the basis of this feature vector and the initial weight vector according to formula (11.1). Since $v = 0$, we get

[13] A single perceptron with n inputs can be used for dividing the n-dimensional *feature space* into two areas corresponding to two classes.

[14] Let us remember that $X_0 = 1$ according to our earlier assumption.

[15] In order to show the *idea* of perceptron learning in a few steps, we make *convenient assumptions*, e.g., that the randomly selected initial weight vector is of such a form.

an output signal $y = f(v) = -1$ according to formulas (11.2) and (11.6). However, the required output signal is $u = 1$ for the character A. Thus, we have to modify the weight vector according to formula (11.7). One can easily check that $\mathbf{W}(2) = (1, 1, 1, 1, 1, 1, 0, 1)$ is the new (modified) weight vector.[16]

Step 2. The character C, i.e., the feature vector $(1, 1, 1, 0, 0, 0, 1, 1)$ is shown to the perceptron. We compute a value v on the basis of this feature vector and the weight vector $\mathbf{W}(2)$. Since $v = 4$, the output signal $y = f(v) = 1$. However, the required output signal $u = -1$ for the character C. Thus, we have to modify the weight vector. It can easily be checked that $\mathbf{W}(3) = (0, 0, 0, 1, 1, 1, -1, 0)$ is the new (modified) weight vector.[17]

Step 3. The character A is shown to the perceptron once again. We compute a value v on the basis of this feature vector and the weight vector $\mathbf{W}(3)$. Since $v = 3$, the output signal $y = f(v) = 1$ which is in accordance with the required signal $u = 1$. Thus, we do not modify the weight vector, i.e., $\mathbf{W}(4) = \mathbf{W}(3)$.

Step 4. The character C is shown to the perceptron once again. We compute a value v on the basis of this feature vector and the weight vector $\mathbf{W}(4)$. Since $v = -1$, the output signal $y = f(v) = -1$ which is in accordance with the required signal $u = 1$. Thus, we do not modify the weight vector, i.e., $\mathbf{W}(5) = \mathbf{W}(4)$.

Step 5. The learning process is complete, because the perceptron recognizes (classifies) both characters in the correct way.

Let us notice that the weight vector obtained as a result of the learning process,

$$\mathbf{W} = (W_0, W_1 = 0, W_2 = 0, W_3 = 1, W_4 = 1, W_5 = 1, W_6 = -1, W_7 = 0),$$

has an interesting interpretation. The *neutral* weights $W_1 = W_2 = W_7 = 0$ mean that features X_1, X_2, and X_7 are the same in both patterns. The *positive* weights $W_3 = W_4 = W_5 = 1$ enhance features X_3, X_4, and X_5, which occur (are switched on) in the pattern A and not in the pattern C. On the other hand, the *negative* weight $W_6 = -1$ weakens the feature X_6, which occurs (is switched on) in the pattern C and not in the pattern A.

Although the perceptron is one of the earliest neural network models, it is still an object of advanced research because of its interesting learning properties [25].

After introducing the basic notions concerning construction, behavior, and learning an artificial neuron, we discuss differences among various types of artificial neurons. A typology of artificial neuron models can be defined according to the following four criteria:

- the structured functional scheme,
- the rule used for learning,
- the kind of the activation function,
- the kind of postsynaptic potential function.

A scheme of a neuron of a certain type which presents its functional components (e.g., an adder computing the value of the postsynaptic potential function,

[16] We add the vector $\mathbf{X}_A$ to the vector $\mathbf{W}(1)$, because $u = 1$.

[17] We subtract the vector $\mathbf{X}_C$ from the vector $\mathbf{W}(2)$, because $u = -1$.

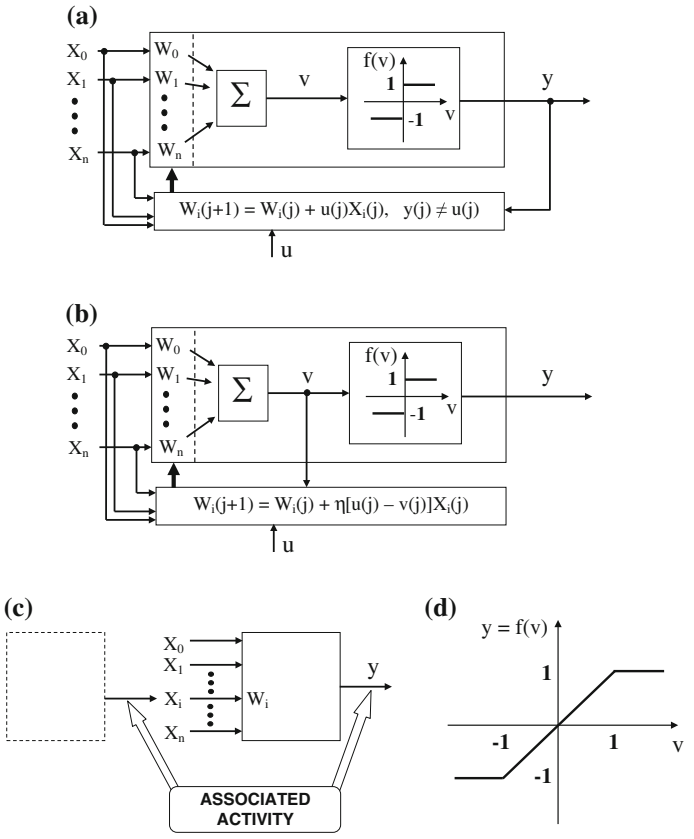

Fig. 11.5 Structured functional schemes of **a** a perceptron and **b** an Adaline neuron; **c** the scheme of Hebb's rule, **d** a piecewise linear activation function

a component generating the value of the activation function, etc.) and data/signal flows among these components is called a *structured functional scheme*.[18] Such a scheme for a perceptron is shown in Fig. 11.5a. A structured functional scheme for an *Adaline* (Adaptive Linear Neuron) introduced by Bernard Widrow[19] and Marcian E. "Ted" Hoff[20] in 1960 [313] is shown in Fig. 11.5b. One can easily notice that in the Adaline scheme the signal v is an input signal of the learning component. (In the perceptron model the signal y is used for learning.)

[18] There is no standard notation for structured functional schemes. Various drawing conventions are used in monographs concerning neural networks.

[19] Bernard Widrow—a professor of electrical engineering at Stanford University. He invented, together with T. Hoff, the least mean square filter algorithm (LMS). His work concerns pattern recognition, adaptive signal processing, and neural networks.

[20] In 1971 Ted Hoff, together with Stanley Mazor, Masatoshi Shima, and Federico Faggin, designed the first microprocessor—Intel 4004.

In the case of these two models of neurons the difference between them is not so big and concerns only the signal flow in the learning process. However, in the case of advanced models, e.g., dynamic neural networks [117], the differences between the schemes can be significant.

Secondly, the neuron models differ from each other regarding the *learning rule*. For example, the learning rule for the perceptron introduced above is defined by formula (11.7), whereas the learning rule for the Adaline neuron is formulated in the following way:

$$W_i(j+1) = W_i(j) + \eta[u(j) - v(j)]X_i(j), \tag{11.8}$$

where $X_i(j)$ is the ith coordinate of the vector shown at the jth step, $u(j)$ is the output signal required for this vector, $v(j)$ is the signal received according to rule (11.1), and η is the learning-rate coefficient. (It is a parameter of the method which is determined experimentally.)

One of the most popular learning rules is based on research led by Donald O. Hebb,[21] which concerned the learning process at a synaptic level. Its results, published in 1949 [133] allowed him to formulate a learning principle called *Hebb's rule*. This is based on the following observation [133]:

> The general idea is an old one, that any two cells or systems of cells that are repeatedly active at the same time will tend to become "associated", so that activity in one facilitates activity in the other.

This relationship is illustrated by Fig. 11.5c (for artificial neurons). Activity of the neuron causes generation of an output signal y. The activity can occur at the same time as activity of a preceding neuron (marked with a dashed line in the figure). According to the observation of Hebb, in such a case neurons become *associated*, i.e., the activity of a *preceding* neuron causes activity of its *successor* neuron. Since the output of the preceding neuron is connected to the X_i input of its successor, the association of the two neurons is obtained by increasing the weight W_i. This relationship can be formulated in the form of *Hebb's rule* of learning:

$$W_i(j+1) = W_i(j) + \eta y(j)X_i(j), \tag{11.9}$$

where η is the learning-rate coefficient. Let us notice that the input signal $X_i(j)$ is equated here with activity of the preceding neuron which causes the generation of its output signal. Of course, both neurons are self-learning, i.e., without the help of a *teacher*.

The *kind of activation function* is the third criterion for defining a taxonomy of neurons. We have already introduced activation functions for the McCulloch-Pitts neuron (the Heaviside step function) and the perceptron (the bipolar step function). Both functions operate in a very *radical* way, i.e., by a step. If we want a less radical operation, we define a *smooth* activation function. For example, the piecewise linear activation function shown in Fig. 11.5d is of the following smooth form:

[21] Donald Olding Hebb—a professor of psychology and neuropsychology at McGill University. His work concerns the influence of neuron functioning on psychological processes such as learning.

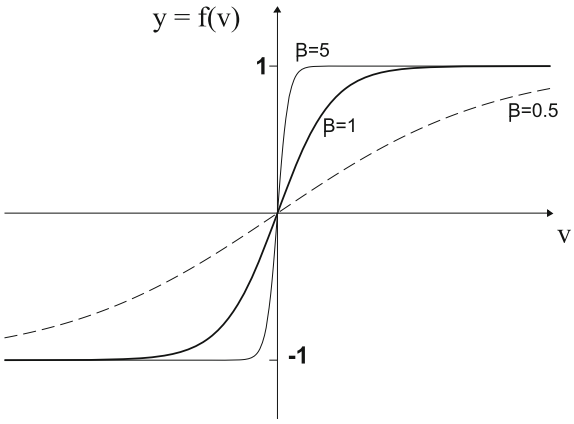

Fig. 11.6 The sigmoidal activation function

$$f(v) = \begin{cases} 1, & \text{if } v > 1, \\ v, & \text{if } -1 \leq v \leq 1, \\ -1, & \text{if } v < -1. \end{cases} \tag{11.10}$$

In the case of a *sigmoidal neuron* we use the *sigmoidal activation function*, which is even smoother (cf. Fig. 11.6). It is defined by the following formula (the bipolar case):

$$f(v) = tanh(\beta v) = \frac{1 - e^{\beta v}}{1 + e^{-\beta v}}. \tag{11.11}$$

As we can see in Fig. 11.6, the greater the value of the parameter β is, the more rapidly the output value changes and the function is more and more similar to the piecewise linear function. In the case of advanced neuron models we use more complex functions such as e.g., radial basis functions or functions based on the Gaussian distribution (we discuss them in Sect. 11.3).

The *kind of postsynaptic potential function* is the last criterion introduced in this section. It is used for computing the total sum of the affecting input signals of the neuron. In this section we have assumed that it is of the form of a sum, i.e., it is defined by formula (11.1). In fact, such a form is used in many models of neurons. However, other forms of postsynaptic potential function can be found in the literature in the case of advanced models such as Radial Basis Function neural networks, fuzzy neural networks, etc.

11.2 Basic Structures of Neural Networks

As we have already mentioned, we use (artificial) neurons for building structures called (artificial) neural networks. A *one-layer neural network* is the simplest structure. Its scheme is shown in Fig. 11.7a. Input signals are usually sent to all neurons of such a structure. In the case of a *multi-layer neural network*, neurons which belong

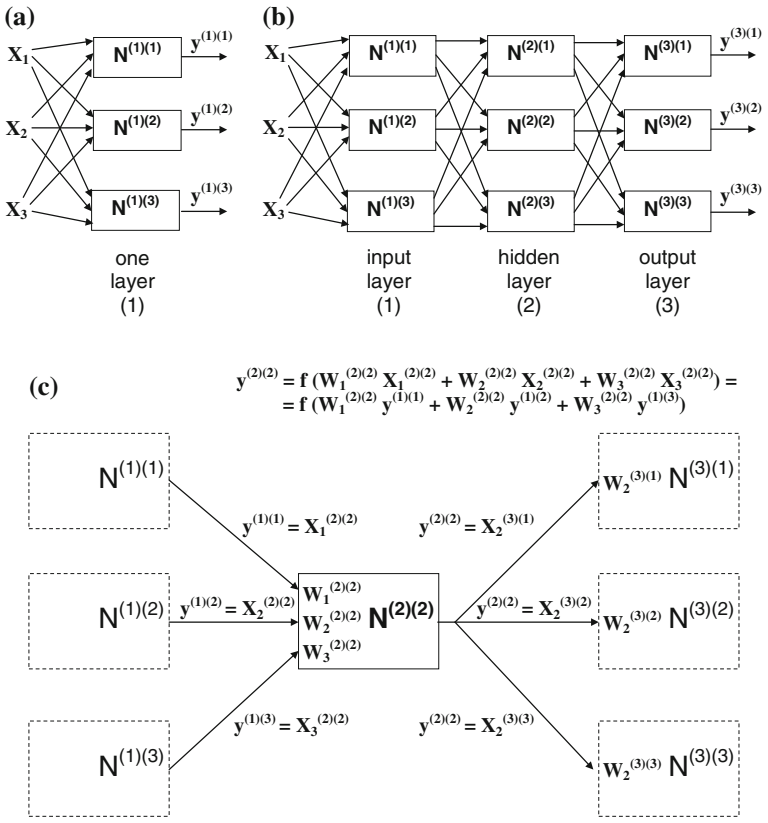

Fig. 11.7 Examples of feedforward neural networks: **a** one-layer, **b** multi-layer (three-layer); **c** the scheme for computing an output signal in a multi-layer network

to the rth layer send signals to neurons which belong to the $(r+1)$th layer. Neurons which belong to the same layer cannot communicate (cf. Fig. 11.7b). The first layer is called the *input layer*, the last layer is called the *output layer*, and intermediate layers are called *hidden layers*. Networks (one-layer, multi-layer) in which signals flow in only one direction are called *feedforward neural networks*. Let us assume the following notations (cf. Fig. 11.7b, c). $N^{(r)(k)}$ denotes the kth neuron of the rth layer, $y^{(r)(k)}$ denotes its output signal. Input signals to the neuron $N^{(r)(k)}$ are denoted by $X_i^{(r)(k)}$, where $i = 1, \ldots, n$, n is the number of inputs to this neuron,[22] and its weights are denoted by $W_i^{(r)(k)}$. Let us notice also (cf. Fig. 11.7b, c) that the following holds:

$$y^{(r-1)(k)} = X_k^{(r)(p)} \tag{11.12}$$

[22] In our considerations we omit the input $X_0^{(r)(k)}$, because it equals 1 and we omit the weight $W_0^{(r)(k)}$, because it can be (as a constant) taken into account by modifying an activation threshold.

for any pth neuron of the rth layer. In other words, for any neuron which belongs to the rth layer its kth input is connected with the output of the kth neuron of the preceding layer, i.e., of the $(r-1)$th layer.[23]

An example of computing the output signal for a neuron of a feedforward network is shown in Fig. 11.7c. As we can see, for a neuron $N^{(2)(2)}$ the output signal is computed according to formulas (11.1) and (11.2), i.e., $y^{(2)(2)} = f(W_1^{(2)(2)} X_1^{(2)(2)} + W_2^{(2)(2)} X_2^{(2)(2)} + W_3^{(2)(2)} X_3^{(2)(2)})$. In the general case, we use the following formula for computing the output signal for a neuron $N^{(r)(k)}$ taking into account the relationship (11.12):

$$y^{(r)(k)} = f(\sum_i W_i^{(r)(k)} X_i^{(r)(k)}) = f(\sum_i W_i^{(r)(k)} y^{(r-1)(i)}). \tag{11.13}$$

The *backpropagation* method was published by David E. Rumelhart, Geoffrey E. Hinton and co-workers in 1986 [252]. It is the basic learning technique of feedforward networks. At the first step output signals for neurons of the output layer L are computed by subsequent applications of formula (11.3) for neurons which belong to successive layers of the network. At the second step we compute errors $\delta^{(L)(k)}$ for every kth neuron of the Lth (output) layer according to the following formula (cf. Fig. 11.8a):

$$\delta^{(L)(k)} = (u^{(k)} - y^{(L)(k)}) \frac{df(v^{(L)(k)})}{dv^{(L)(k)}}, \tag{11.14}$$

where $u^{(k)}$ is the correct (required) output signal for the kth neuron of the Lth layer and f is the activation function.

Then, errors of neurons of the given layer are *propagated backwards* to neurons of a preceding layer according to the following formula (cf. Fig. 11.8b):

$$\delta^{(r)(k)} = \sum_m (\delta^{(r+1)(m)} W_k^{(r+1)(m)}) \frac{df(v^{(r)(k)})}{dv^{(r)(k)}}, \tag{11.15}$$

where m goes through the set of neurons of the $(r+1)$th layer. In the last step we compute new weights $W_i^{'(r)(k)}$ for each neuron $N^{(r)(k)}$ on the basis of the computed errors in the following way (cf. Fig. 11.8c):

$$W_i^{'(r)(k)} = W_i^{(r)(k)} + \eta \delta^{(r)(i)} X_i^{(r)(k)} = W_i^{(r)(k)} + \eta \delta^{(r)(i)} y^{(r-1)(i)}, \tag{11.16}$$

where η is the learning-rate coefficient. A mathematical model of learning with the help of the backpropagation method and the derivation of formulas (11.14)–(11.16) are contained in Appendix H.

Fundamental problems of neural network learning include determining a stopping condition for a learning process, how to compute the error of learning, determining

[23] Thus, we could omit an index of the neuron in the case of input signals, retaining only the index of the layer, i.e., we could write $X_i^{(r)}$ instead of $X_i^{(r)(k)}$.

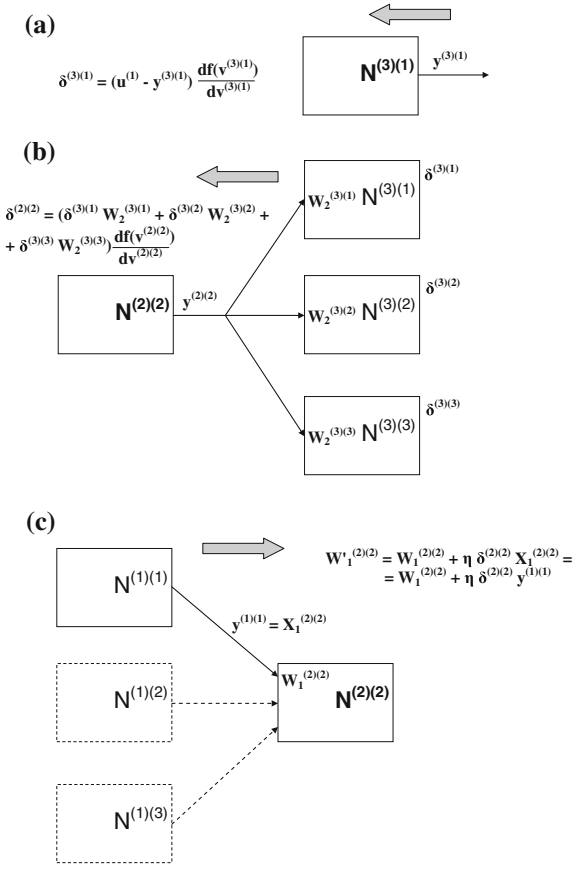

Fig. 11.8 Backpropagation learning: **a** computing the error for a neuron of the output layer, **b** error propagation, **c** computing a weight

initial weights, and speeding up the learning process. We do not discuss these issues in the monograph, because of its introductory nature. The reader is referred to monographs included in the bibliographical note at the end of this chapter.

A neural network can be designed in such a way that it contains connections from some layers to preceding layers. In such a case input signals can be propagated from later processing phases to earlier phases. Such networks are called *recurrent neural networks*, from Latin *recurrere–running back*. They have great computing power that is equal to the computing power of a Turing machine [156]. The first recurrent neural network was introduced by John Hopfield in 1982 [140]. Neurons of a *Hopfield network* are connected as shown in Fig. 11.9 (for the case of three neurons).[24] Input signals are directed multiply to the input of the network and signals

[24]We usually assume, following the first paper of Hopfield [140], that neuron connections are symmetrical, i.e., $W_i^{(1)(k)} = W_k^{(1)(i)}$. However, in generalized models of Hopfield networks this property is not assumed.

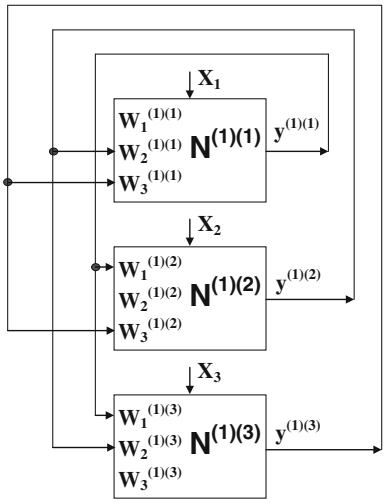

Fig. 11.9 A recurrent Hopfield network

recur for some time until the system stabilizes. Hopfield networks are also used as associative memory networks, which are introduced in the next section.

In general, recurrent networks can be multi-layer networks. In 1986 Michael I. Jordan[25] proposed a model, then called the *Jordan network* [151]. In this network, apart from an input layer, a hidden layer and an output layer, an additional *state layer* occurs. Inputs of neurons of the state layer are connected to outputs of neurons of the output layer and outputs of neurons of the state layer are connected to inputs of neurons of the hidden layer. Jordan networks are used for modeling human motor control.

A similar functional structure occurs in the *Elman network* defined in the 1990s by Jeffrey L. Elman[26] [84]. The main difference of this model with respect to the Jordan network consists of connecting inputs of neurons of an additional layer, called here the *context layer*, with outputs of neurons of the hidden layer (not the output layer). However, outputs of neurons of the context layer are connected to inputs of neurons of the hidden layer, as in the Jordan model. Elman networks are used in Natural Language Processing (NLP), psychology, and physics.

11.3 Concise Survey of Neural Network Models

This section includes a concise survey of the most popular models of neural networks.

Autoassociative memory networks are used for storing pattern vectors in order to recognize similar patterns with the help of an association process. They can be

[25] Michael Irwin Jordan—a professor of computer science and statistics at the University of California, Berkeley and the Massachusetts Institute of Technology. His achievements concern self-learning systems, Bayesian networks, and statistical models in AI.

[26] Jeffrey Locke Elman—an eminent psycholinguist, a professor of cognitive science at the University of California, San Diego.

used for modeling associative storage[27] in computer science. Processing incomplete information is their second important application field. In this case the network simulates one of the fundamental functionalities of the brain, that is the ability to restore a complete information on the basis of incomplete or distorted patterns with the help of an association process.[28] The original research into autoassociative memory networks was led in the early 1970s by Teuveo Kohonen [164]. The most popular neural networks of this type include the following models[29]:

- A two-layer, feedforward, with supervised learning *Hinton network*, which was introduced by Geoffrey Hinton in 1981 [136].
- A *bidirectional associative memory, BAM*, network proposed by Stephen Grossberg and Michael A. Cohen[30] [54].[31] The BAM model can be considered to be a generalization of a (one-layer) Hopfield network for a two-layer recurrent network. Signal flows occur in one direction and then the other in alternate cycles (*bidirectionally*) until the system stabilizes.
- *Hamming networks* were introduced by Richard P. Lippmann[32] in 1987 [183]. They can also be considered to be a generalization of Hopfield networks with a three-layer recurrent structure. Input and output layers are feedforward and the hidden layer is recurrent. Their processing is based on minimizing the Hamming distance[33] between an input vector and model vectors stored in the network.[34]

Self-Organizing Maps, SOMs, were introduced by Teuvo Kohonen [165] in 1982. They are used for cluster analysis, discussed in Sect. 10.7. Kohonen networks generate a discrete representation called a *map* of low dimensionality (maps are usually two- or three-dimensional) on the basis of elements of a learning set.[35] The map shows *clusters* of vectors belonging to the learning set. In the case of Self-Organizing Maps we use a specific type of unsupervised learning, which is called *competitive learning*.

---

[27] Associative storage allows a processor to perform high-speed data search.

[28] For example, if somebody mumbles, we can *guess* correct words. If we see a building that is partially obscured by a tree, we can *restore* a view of the whole building.

[29] Apart from the Hopfield networks introduced in the previous section.

[30] Michael A. Cohen—a professor of computer science at Boston University, Ph.D. in psychology. His work concerns Natural Language Processing, neural networks, and dynamical systems.

[31] The BAM model was developed significantly by Bart Kosko [170], who has been mentioned in Chap. 1.

[32] Richard P. Lippmann—an eminent researcher at the Massachusetts Institute of Technology. His work concerns speech recognition, signal processing, neural networks, and statistical pattern recognition.

[33] The Hamming metric is introduced in Appendix G.

[34] Lippmann called the model the Hamming network in honor of Richard Wesley Hamming, an eminent mathematician whose works influenced the development of computer science. Professor Hamming programmed the earliest computers in the Manhattan Project (the production of the first atomic bomb) in 1945. Then, he collaborated with Claude E. Shannon at the Bell Telephone Laboratories. Professor Hamming has been a founder and a president of the Association for Computing Machinery.

[35] This set can be defined formally by (11.15).

Output neurons compete in order to be activated during the process of showing patterns. Only the best neuron, called the *winner*, is activated. The *winner* is the neuron for which the distance between its weight vector and the shown vector is minimal. Then, in the case of competitive learning based on a *WTA* (*Winner Takes All*) strategy only the weights of the winner are modified. In the case of a *WTM* (*Winner Takes Most*) strategy weights are modified not only for the winner, but also for its neighbors ("the winner takes most [of a prize], but not all").

ART (Adaptive Resonance Theory) neural networks are used for solving problems of pattern recognition. They were introduced by Stephen Grossberg and Gail Carpenter [42] for solving one of the most crucial problems of neural network learning, namely to increase the number of elements of a learning set. In case we increase the number of elements of the learning set,[36] the learning process has to start from the very beginning, i.e., including patterns which have already been shown to the network. Otherwise, the network could *forget* about them. Learning, which corresponds to cluster analysis introduced in Sect. 10.7, is performed in ART networks in the following way. If a new pattern is *similar* to patterns belonging to a certain class, then it is added to this class. However, if it is not *similar* to any class, then it is not added to the *nearest* class, but a new class is created for this pattern. Such a strategy allows the network to preserve characteristics of the classes defined so far. For the learning process a *vigilance parameter* is defined, which allows us to control the creation of new classes. With the help of this parameter, we can divide a learning set into a variety of classes which do not differ from each other significantly or we can divide it into a few generalized classes.

Probabilistic neural networks defined by Donald F. Specht in 1990 [283] recognize patterns on the basis of probability density functions of classes in an analogous way to statistical pattern recognition introduced in Sect. 10.5.

Boltzmann machines[37] can be viewed as a precursor of probabilistic neural networks. They were defined by Geoffrey E. Hinton and Terrence J. Sejnowski[38] in 1986 [137].

Radial Basis Function, RBF, neural networks are a very interesting model. In this model activation functions in the form of *radial basis functions*[39] are defined for each neuron separately, instead of using one global activation function. If we use a *standard* neural network with one activation function (the step function, the sigmoidal function, etc.), then we divide the feature space into subspaces (corresponding to classes) in a *global* way with the help of all the neurons which take part in the process. This is consistent with the idea of a *distributed connectionist network model* introduced in

[36] Of course, this is recommended, since the network becomes "more experienced".

[37] Named after a way of defining a probability according to the Boltzmann distribution, similarly to the *simulated annealing* method introduced in Chap. 4.

[38] Terrence "Terry" Joseph Sejnowski—a professor of biology and computer science and director of the Institute of Neural Computation at the University of California, San Diego (earlier at California Institute of Technology and John Hopkins University). John Hopfield was an adviser of his Ph.D. in physics. His work concerns computational neuroscience.

[39] The value of a radial basis function depends only on the distance from a certain point called the *center*. For example, the Gaussian function can be used as a radial basis function.

Sect. 3.1. Meanwhile, the behavior of the basis function changes around the center radially. It allows a neuron to "isolate" (determine) a subarea of the feature space in a *local* way. Therefore, RBF networks are sometimes called *local networks*, which is a reference to the *local connectionist network model* introduced in Sect. 3.1.

Bibliographical Note

There are a lot of monographs on neural networks. Books [27, 50, 87, 129, 257, 324] are good introductions to this area of Artificial Intelligence.