2 Neuron

learning objectives

- neural networks in biology; neurons and synapses
- inputs, weights, outputs, and input-output conversion (transfer) functions
- linear learning machine
- delta-rule
- graphical representation of artificial neurons

2.1 Synapses and Input Signals

Whether artificial or computer simulated, a "neuron" is designed to mimic the function of the neural cells of a living organism. Therefore, we should look at a brief description of the biological neuron. For further details, see textbooks on physiology or neurophysiology (e.g. Color Atlas of Physiology).

The human nerve system consists of about 10^{10} neural cells, also called neurons. Although there are at least five different types of neural cells, it suffices to present only one type. A typical *neuron* of the motor complex consists of a cell body (soma) with a nucleus. The cell body has two types of extensions: the dendrites and the axon. Figure 2-1 shows a drastically simplified picture of such a neuron. The dendrites receive signals and send them to the soma. A neuron has substantially more dendrites than indicated in Figure 2-1, which are also much more branched. Thus, the dendrites have quite a large surface (up to 0.25 mm^2) available to receive signals from other neurons. The axon, which transmits signals to other neurons (or to muscle cells), branches into several "collaterals".

The axons and collaterals end in *synapses*. These synapses make contact with the dendrites or the somata of other neurons. A motor neuron has thousands of synapses; up to 40% of the surface of a neuron is covered with such contact sites.

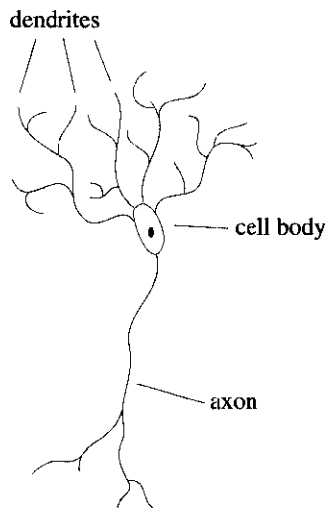

Figure 2-1: Overly simplified scheme of a motor neuron.

The transfer of signals within the dendrites and the axon is electrical, occurring through the transport of ions. However, the signal is transmitted across the synapse by chemical substances. The electric signal in the axon releases a chemical substance, the *neurotransmitter* (for example, acetylcholine), which is stored in vesicles at the presynaptic membrane. This neurotransmitter diffuses across the synaptic gap and through the postsynaptic membrane into the dendrite of the other neuron (Figure 2-2).

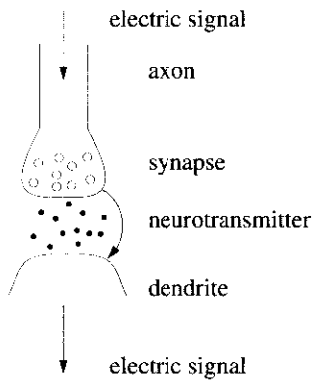

Figure 2-2: Schematic representation of a synapse.

In the dendrite, the neurotransmitter generates a new electric signal that is passed through the second neuron. Since the postsynaptic membrane cannot release the neurotransmitter, the synapses can only send the signal in one direction, and therefore function as gates; this is an essential prerequisite for the transmission of information. In addition, other neurons can **modify** the transmission of signals at the synapses.

The signals produced by the neurons, regardless of the species producing them, are very similar and therefore almost indistinguishable, even when produced by a very primitive or a highly sophisticated (from the evolutionary point of view) species. Kuffler and Nicholls say in their book "From Neuron to Brain" (page 4):
"... these signals are virtually identical in all nerve cells of the body ... [and] are so similar in different animals that even a sophisticated investigator is unable to tell with certainty whether a photographic record of a nerve impulse is derived from the nerve fibre of a whale, mouse, monkey, worm, tarantula, or professor."

To be very clear, the intensity of signals produced by the neurons (the frequency of firing) can differ depending on the intensity of the stimulus. However, the shape and overall appearance of different signals are very similar.

Why is this conclusion of neural network research so important? A short (if not the most comprehensive) answer to this question is that the similarity of signals clearly suggests that the real functioning of the brain is not so much dependent on the role of a single neuron, but rather on the entire ensemble of neurons – that is, the way the neurons are interconnected.

> Therefore, the emphasis in the phrase "neural network" is on "network" rather than on "neural".

The synapses, through which the signals from neighboring neurons enter into one particular neuron, represent barriers which will almost certainly modulate a signal passing through them. The amount of change depends on the so called *synaptic strength*. In artificial neurons, the synaptic strength is called a *weight*, *w*. This situation is shown schematically in Figure 2-3.

Without going into the physics and chemistry of membranes, we can say that the synaptic strength determines the relative amount of the signal that enters the body of the neuron through the dendrites. Fast changes of the synaptic strengths, even between two consecutive impulses, are regarded as a vital mechanism in the proper and efficient functioning of the brain. The adaptation of synaptic strengths to a particular problem is the essence of learning.

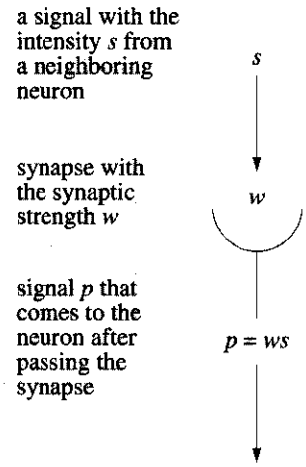

Figure 2-3: The *synaptic strength w* changes the intensity of the incoming signal *s*.

2.2 Weights

Because each neuron has a large number of dendrites/synapses (Figure 2-1), many signals can be received by the neuron simultaneously. The individual signals are labelled s_i and the corresponding synaptic strengths (weights), w_i. Assuming that the weight at each of the neuron's numerous synapses can have a different value at a given moment, we can estimate that the incoming signals can add together into a kind of *collective effect*, or *net input*.

In reality we do not know exactly how the collective effect is formed, nor do we know how large it is with respect to all input signals. Therefore, some very crude simplifications must be made when making a model of a neuron:

- the net input (called *Net*) is a function of all signals s_i that arrive within a given time interval, and of all synaptic strengths (weights, w_i); and
- the function linking these quantities is a simple sum of products of the entering signals s_i and the corresponding weights w_i. Thus, we can write:

$$Net = w_1 s_1 + w_2 s_2 + \ldots + w_i s_i + \ldots + w_m s_m \qquad (2.1)$$

This is not the only way to represent the net input of the neuron; some authors have proposed quite elaborate functions. However, it is

believed that the representation of *Net* may be relatively simple if we intend to simulate a **large** assembly of neurons.

Some attentive readers may have noticed that until now we have avoided talking about the "output" of the neuron. Since we do not know what happens inside the neuron, we can not say to what extent the net input is equal to the real output of the sending neuron, and to what extent it is modified by the receiving one. However, for reasons that will become clear later, the present model calculates the actual output of the neuron in two steps. At the moment, we are only concerned with the **first step** of this calculation, the evaluation of the so called net input *Net*!

This simple calculation using Equation (2.1) corresponds to the schematic of Figure 2-4. The representation of the artificial neuron is inspired by the structure of a real neuron.

As an example, we will now calculate the net input *Net* of a neuron having only four synapses with weights 0.1, 0.2, –0.3 and –0.02. Because we have designed our artificial neuron to have only four synapses, it can handle signals from only **four** contacting neurons simultaneously (in this example, signals with intensities 0.7, 0.5, –0.1, 1.0 – see Figure 2-5). Because the net input *Net* can not be identified with the real output of the neuron, only the upper half of a circle representing the neuron's body is drawn. The lower half of this circle (dashed line) reminds us that another step is needed to obtain the output from the net input. This will be explained in Section 2.4.

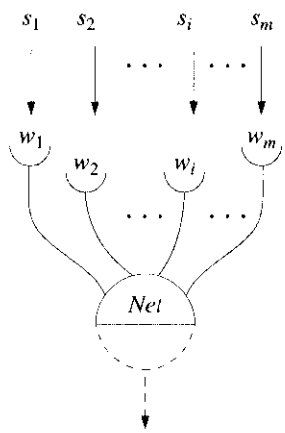

Figure 2-4: Calculation of the net input *Net* to an artificial neuron having *m* synapses.

As can be seen from this example, the net input of an **artificial** neuron can have negative or positive values; because there is no reason for limiting either the signs or the magnitudes of the weights, the net inputs can have a very large range of values. This is especially true if neurons with thousands of synaptic connections to surrounding neurons are taken into account.

Rather than a group of signals $s_1, s_2, s_3, ..., s_i, ..., s_m$ received by the given neuron from the surrounding m neurons, it is much more convenient to combine them into a multivariate signal: a multi-dimensional vector $\boldsymbol{X}$, whose components are the individual signals:

$$\{ s_1, s_2, s_3, ..., s_i, ..., s_m \} = \boldsymbol{X}(x_1, x_2, ..., x_m) \qquad (2.2)$$

Using this notation, the 4-dimensional input vector $\boldsymbol{X}$ for our previous example would be written as:

$$\boldsymbol{X} = (\, 0.7,\ 0.5,\ -0.1,\ 1.0\,) \qquad (2.3)$$

When we use the same line of reasoning, all the synaptic strengths in a neuron can be described by using a multidimensional *weight vector* $\boldsymbol{W}$:

$$\boldsymbol{W} = (w_1, w_2, w_3, ..., w_i, ..., w_m) \tag{2.4}$$

With only one neuron, the 4-dimensional weight vector of Figure 2-5 will look like this:

$$\boldsymbol{W} = (0.1,\ 0.2,\ -0.3,\ -0.02) \tag{2.5}$$

It is evident that each neuron should have at least as many weights as attached neurons. In a biological neuron, a synapse with a certain synaptic strength is immediately formed when an axon and a dendrite link together; in a computer simulated one, the programmer has to accommodate the corresponding number of weights.

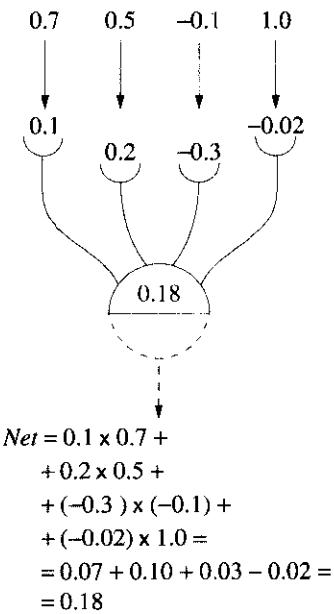

Figure 2-5: Calculation of the net input of a neuron with four synapses.

2.3 Linear Learning Machine

The so-called *linear learning machine*, a topic found in standard textbooks on pattern recognition methods, introduces many valuable concepts and techniques that may be used in more complicated ways later.

The linear learning machine, employing a linear transformation (represented by Equation (2.1)) on a multivariate signal $\boldsymbol{X}$ using the weight vector $\boldsymbol{W}$ to obtain a one-variable (univariate) signal, was very popular in the sixities in many of the sciences including chemistry (see Nilsson: Linear Learning Machines).

The learning machine procedure is mainly used for deciding whether a given multivariate input signal $\boldsymbol{X}$ belongs to a certain category (see Jurs and Isenhour, *Analytical Chemistry*, August 1971; cf. Figure 2-10).

Such a multidimensional vector $\boldsymbol{X}$ can represent many things, such as:

- an audio signal,
- an optical image,
- a spectrum of any kind,
- a many-component chemical analysis,
- the status of a technological process at a given time,

- the sequence of amino acids in a protein,
- weather records for a certain location,
- the market values of some stocks.

Hence, the possibility of finding an appropriate weight vector $\boldsymbol{W}$ that would produce correct decisions is very attractive.

The goal of the linear learning machine was to input a set of m-dimensional signals $\boldsymbol{X}$, and find the appropriate vector $\boldsymbol{W}$ by slowly improving an initial guess, $\boldsymbol{W}^{(0)}$, through a number of corrections obtained by comparing the result with the correct decisions known in advance. In order to maintain similarity with the neural network, the result of this procedure is here called *Net*. Equation (2.6):

$$Net = w_1 s_1 + w_2 s_2 + \ldots + w_i s_i + \ldots + w_m s_m \qquad (2.6)$$

can be regarded as the dot product of two vectors: the weight vector $\boldsymbol{W}$ and the vector $\boldsymbol{X}$ containing m signals s_i. Because we prefer vector notation to this extended one, the individual signals s_i are labeled as the components x_i of a multisignal input vector $\boldsymbol{X}$:

$$Net = w_1 x_1 + w_2 x_2 + \ldots + w_i x_i + \ldots + w_m x_m = \boldsymbol{WX} \qquad (2.7)$$

The input vector is linearly proportional to the corrected result, *Net*; hence, the corrective procedure (2.7) is called a **linear** learning machine.

Now, the "input vector $\boldsymbol{X}$" is a set of signals: a multivariate object described by several parameters. In case there are more objects for input, they are distinguished by an index s, e.g. $\boldsymbol{X}_s$. Therefore, the actual univariate signals coming to the individual synapses are components of these multivariate objects having two indices x_{si}, the first of which labels the multivariate object and the second of which labels the synapse to which this individual signal is linked.

From now on the components x_{si} will be called *signals*, and the multivariate inputs will be called *objects* or *vectors* $\boldsymbol{X}_s$. The multivariate space, where the objects are represented as radius vectors leading to points $\boldsymbol{X}_s$, is called the *measurement space*.

In Equations (2.6) and (2.7), the terms *Net*, $\boldsymbol{W}$ and $\boldsymbol{X}$ are so named to remind us of the corresponding features in the artificial neuron.

Net, the scalar product of a weight vector $\boldsymbol{W}$ and a multivariate vector $\boldsymbol{X}_s$ representing an arbitrary object in measurement space, is a very convenient quantity for making decisions. Its sign can indicate to

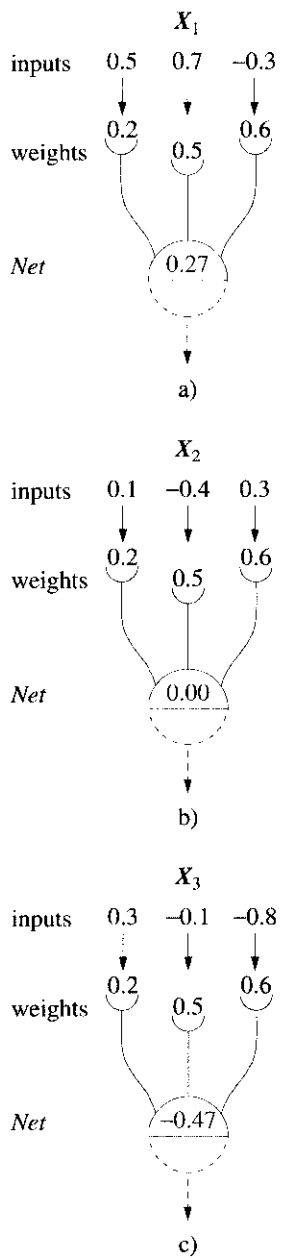

Figure 2-6: Dot products *Net* between the weight vector $\boldsymbol{W} = (0.2, 0.5, 0.6)$ and three arbitrarily chosen input vectors $\boldsymbol{X}_1$, $\boldsymbol{X}_2$ and $\boldsymbol{X}_3$.

which of the two categories selected in advance, C_1 or C_2, the object X_s belongs.

Figure 2-6 shows how a vector W (0.2, 0.5, 0.6) can be used as a decision vector to classify three different 3-dimensional objects X_1 = (0.5, 0.7, –0.3), X_2 = (0.1, –0.4, 0.3), X_3 = (0.3, –0.1, –0.8).

Separating objects into classes based on whether *Net* is positive or negative is not the only possible division; *Net* can be easily divided into three or more intervals to define three or more categories. For example, a division into the intervals $-\infty$ to $-a$, $-a$ to $+a$, and $+a$ to $+\infty$ can be used to decide to which of the three categories object $\mathbf{X}_s$ belongs. Most of the decisions are binary, i.e. decisions between two categories, because all complex decisions can be composed of a series of binary ones.

Let the weight vector W be selected so that all objects X_s belonging to category C_1 give the scalar product $Net = WX_s$ positive, and all objects X_s from C_2 give $Net = WX_s$ negative; then W can be called a *perfect decision vector.* In principle, a perfect decision vector can be found if the objects are linearly separable. However, practical ways of obtaining a good (let alone perfect!) decision vector are hard to come by, if the data are not linearly separable.

Usually W is obtained from an initial (presumably bad) guess $W^{(0)}$ by some kind of learning procedure. $W^{(0)}$ is improved iteratively; an iterative procedure in this context means that if $W^{(t+1)}$ is obtained from $W^{(t)}$, then $W^{(t+1)}$ should be a slightly better decision function than $W^{(t)}$ (Figure 2-7). In the literature there are many different learning procedures in addition to the linear learning machine. Some of these from the field of neural networks will be explained later on in this book.

Figure 2-8 shows three decision vectors W_1, W_2, W_3 used in a hierarchical manner to decide in which quadrant of the xy-plane an input point is located.

If we could design a method for producing reliable decision vectors W, decision schemes of any complexity and size could be built from them, making binary decisions (the so-called piecewise linear classifier). The tree of Figure 2-8 is composed of three decisions: whether the point lies to the left or right of the ordinate axis, and (for each of these cases) where it lies with respect to the abscissa. In this case, the objects are points in the xy-plane, and the weight vectors are given in parentheses at the decision nodes of the tree. In order to make

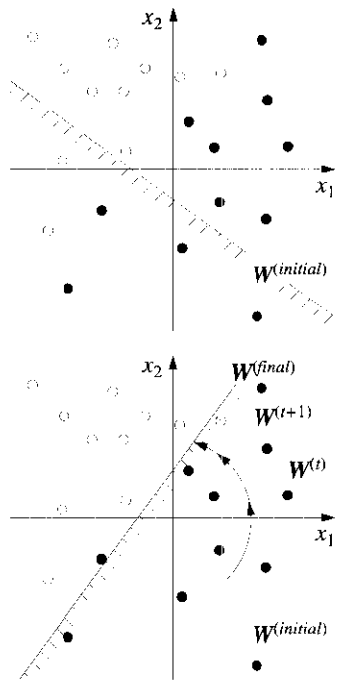

Figure 2-7: Changing the decision vector W towards a better position: $W^{(t+1)}$ is a better decision vector than $W^{(t)}$ because $W^{(t+1)}$ classifies only two objects falsely, compared with five by $W^{(t)}$. The perfect decision vector W should separate **all** objects of category C_1 (full circles) from those of category C_2 (empty circles).

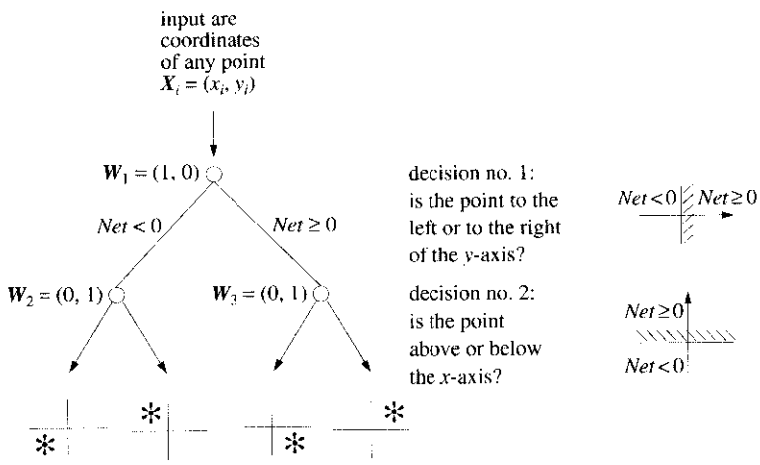

Figure 2-8: A small decision hierarchy for deciding in which quadrant a given point lies.

a decision tree work properly, each binary decision (at any point in the tree) must be **produced** and **checked** separately.

Try to calculate the decisions for some points. For example: take the point $\boldsymbol{P} = (-3, 4)$ and make its dot product *Net* with the weight vector $\boldsymbol{W}_1$: $Net = \boldsymbol{P}\boldsymbol{W}_1 = (-3, 4)(1, 0) = -3$. The result of *Net* determines which branch of the tree in Figure 2-8 has to be followed; in this case, Net < 0 gives the left-hand branch. Then a decision has to be made on the second level. Here, repeat the procedure with $\boldsymbol{W}_2$ to get the final result.

As mentioned above, we can take the decision function (whose result, remember, is *Net*), to be the dot product between the representation of the object $\boldsymbol{X}$ and the weight vector $\boldsymbol{W}$. The dot product between two vectors can be regarded as a linear function:

$$y = ax + b \tag{2.8}$$

where y is equal to *Net*, a and x are the magnitudes of the vectors $\boldsymbol{W}$ and $\boldsymbol{X}$, and b is a scalar constant that can be regarded as the offset of this straight line.

For compatibility with the notation of neural networks, we label this offset parameter as ϑ:

$$Net = \boldsymbol{W}\boldsymbol{X} + \vartheta = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_mx_m + \vartheta = \\ = \sum_{i=1}^{m} w_ix_i + \vartheta \tag{2.9}$$

Clearly, both vectors **W** and **X** must have the same number of components.

Observe the similarity of Equations (2.9) and (2.6); they differ only in the constant ϑ, which actually generalizes the linear form. Because we will meet this constant while discussing the neuron's transfer function (see Section 2.4), let us explain this in more detail.

Because ϑ is an arbitrary scalar constant, it can be written as a product of two other constants, say w_{m+1} and 1. If the two scalar constants are treated as the $(m+1)$-th components of the two vectors $\boldsymbol{W}$ and $\boldsymbol{X}$, Equation (2.9) can be rewritten as a scalar product of these two new $(m+1)$-dimensional vectors. $\boldsymbol{X}$ is identical to the old representation of the object augmented by one dimension, the value of which is always equal to 1 (the so-called augmented feature vector), while w_{m+1} is equal to ϑ.

$$\begin{aligned} Net &= w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_mx_m + \vartheta = \\ &= w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_mx_m + w_{m+1} \cdot 1 = \\ &= \boldsymbol{WX} \end{aligned} \tag{2.10}$$

> From now on, remember that both $\boldsymbol{W}$ and $\boldsymbol{X}$ are $(m+1)$-dimensional.

This new component w_{m+1} has far-reaching significance in neural network learning and adaptation (see Section 2.5 describing the concept of "bias").

As before, the sign of the calculated scalar product *Net* defines the category to which the object $\boldsymbol{X}$ belongs. If the dot product $\boldsymbol{WX}$ for a certain vector $\boldsymbol{X}$ has the wrong sign with respect to the category of the $\boldsymbol{X}$'s, we calculate an increment for $\boldsymbol{W}$ as the difference between some new, corrected weight vector $\boldsymbol{W}^{(new)}$ and the old, uncorrected vector $\boldsymbol{W}^{(old)}$:

$$\Delta \boldsymbol{W} = \boldsymbol{W}^{(new)} - \boldsymbol{W}^{(old)} \tag{2.11}$$

Since the correction of weights will be done in iterative steps (0, 1, 2, ..., t, $t+1$, ...), $\boldsymbol{W}^{(new)}$ and $\boldsymbol{W}^{(old)}$ actually refer to $\boldsymbol{W}^{(t+1)}$ and $\boldsymbol{W}^{(t)}$, respectively.

In order to achieve this correction we will use the so called *delta-rule*, which states that, in order to improve the decision vector, the

correction $\Delta \boldsymbol{W}$ should be proportional to a certain parameter δ, (which is proportional to the error) **and** to the input $\boldsymbol{X}$ for which the wrong answer was obtained. After correction, the new weight vector should classify the vector $\boldsymbol{X}$ **correctly** or at least with a smaller error than before.

$$\Delta \boldsymbol{W} \sim \delta \boldsymbol{X}$$

or:

$$\delta \sim \Delta \boldsymbol{W} / \boldsymbol{X} \tag{2.12}$$

or:

$$\delta = \eta \, (\Delta \boldsymbol{W} / \boldsymbol{X})$$

where δ is the correction constant we are looking for, η is a constant of proportionality and $\boldsymbol{X}$ is the input object **wrongly** classified by $\boldsymbol{W}^{(old)}$. Our goal is to find a parameter δ with which the new weight vector $\boldsymbol{W}^{(new)}$ will classify $\boldsymbol{X}$ correctly.

If the dot product $Net = \boldsymbol{W}^{(old)}\boldsymbol{X}$ has a wrong sign, the dot product $Net = \boldsymbol{W}^{(new)}\boldsymbol{X}$ must have the opposite sign:

$$\boldsymbol{W}^{(new)}\boldsymbol{X} = -\boldsymbol{W}^{(old)}\boldsymbol{X} \tag{2.13}$$

If (2.11) is substituted into (2.12):

$$\delta = \eta \left(\boldsymbol{W}^{(new)} - \boldsymbol{W}^{(old)} \right) / \boldsymbol{X} \tag{2.14}$$

and if the right side of (2.14) is multiplied by $\boldsymbol{X}/\boldsymbol{X}$, we obtain:

$$\delta = \eta \left(\boldsymbol{W}^{(new)} - \boldsymbol{W}^{(old)} \right) \boldsymbol{X} / (\boldsymbol{X} \cdot \boldsymbol{X}) \tag{2.15}$$

In the denominator of (2.15) we obtained the dot product of the vector $\boldsymbol{X}$ with itself:

$$\boldsymbol{X} \cdot \boldsymbol{X} = \sum_{i=1}^{m+1} x_i^2 \tag{2.15a}$$

The expression (2.15a) is called the **norm** of the vector $\boldsymbol{X}$ and is written as $\|\boldsymbol{X}\|^2$. So the next expression is:

$$\delta = \eta \left(\boldsymbol{W}^{(new)} - \boldsymbol{W}^{(old)} \right) \boldsymbol{X} / \|\boldsymbol{X}\|^2 \tag{2.16}$$

This can be written as:

$$\delta = \eta \left(\boldsymbol{W}^{(new)}\boldsymbol{X} - \boldsymbol{W}^{(old)}\boldsymbol{X} \right) / \|\boldsymbol{X}\|^2 \tag{2.17}$$

from which, by substitution of (2.13), the final expression for the correction δ is obtained:

$$\begin{aligned}\delta &= \eta\Big(-\boldsymbol{W}^{(old)}\boldsymbol{X} - \boldsymbol{W}^{(old)}\boldsymbol{X}\Big)/\|\boldsymbol{X}\|^2 \\ &= -2\eta\boldsymbol{W}^{(old)}\boldsymbol{X}/\|\boldsymbol{X}\|^2\end{aligned} \tag{2.18}$$

Using Equation (2.18), the correction $\Delta\boldsymbol{W} = \boldsymbol{W}^{(new)} - \boldsymbol{W}^{(old)}$ can now be easily obtained:

$$\Delta\boldsymbol{W} = \boldsymbol{W}^{(new)} - \boldsymbol{W}^{(old)} = -\Big(2\eta\boldsymbol{W}^{(old)}\boldsymbol{X}/\|\boldsymbol{X}\|^2\Big)\boldsymbol{X} \tag{2.19}$$

or in a more extended form:

$$\boldsymbol{W}^{(new)} = \boldsymbol{W}^{(old)} - \Big(2\eta\boldsymbol{W}^{(old)}\boldsymbol{X}/\|\boldsymbol{X}\|^2\Big)\boldsymbol{X} \tag{2.20}$$

The *delta-rule correction* of $\boldsymbol{W}^{(old)}$, (2.19) and (2.20), **guarantees** that $\boldsymbol{W}^{(new)}$ will classify the object $\boldsymbol{X}$ correctly. However, the delta-rule does not say anything about the other objects that were also classified using $\boldsymbol{W}^{(old)}$; some or all of them might now be classified falsely.

If the proportionality constant η is set to 1, Equation (2.20) gives the corrected decision vector $\boldsymbol{W}^{(new)}$ as the mirror image of $\boldsymbol{W}^{(old)}$ (Figure 2-9). Under certain circumstances, the corrections produced by mirroring are small, but often the mirror image can change $\boldsymbol{W}^{(old)}$ considerably. Such large changes are not desirable, especially at the end of iterative learning, because large changes in the decision vector $\boldsymbol{W}$ mean that a number of previously correctly classified objects might become classified falsely. Therefore, an adaptable correction can sometimes be more appropriate.

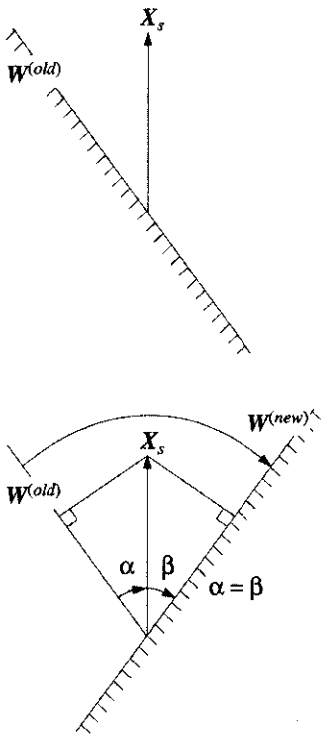

Figure 2-9: The vector $\boldsymbol{W}^{(new)}$, which is the mirror image of $\boldsymbol{W}^{(old)}$, classifies the object $\boldsymbol{X}$ correctly into another category.

An adaptable correction is obtained when, in Equations (2.18) to (2.20), the **constant** η is replaced by a **variable** η, smaller than 1:

$$\boldsymbol{W}^{(new)} = \boldsymbol{W}^{(old)} - \Big(2\eta\boldsymbol{W}^{(old)}\boldsymbol{X}/\|\boldsymbol{X}\|^2\Big)\boldsymbol{X}$$

or: (2.21)

$$\Delta\boldsymbol{W} = -\Big(2\eta\boldsymbol{W}^{(old)}\boldsymbol{X}/\|\boldsymbol{X}\|^2\Big)\boldsymbol{X}$$

Writing the correction $-2\boldsymbol{W}^{(old)}\boldsymbol{X}/\|\boldsymbol{X}\|^2$ as δ and remembering that $\boldsymbol{X}$ is an input object, we obtain the standard equation of the delta-rule in its most widely known form:

$$\Delta\boldsymbol{W} = \eta\ \delta\ \boldsymbol{X} \tag{2.22}$$

Because the object $\boldsymbol{X}$ is the input it can also be called ***Inp***:

$$\Delta\boldsymbol{W} = \eta\ \delta\ \boldsymbol{Inp} \tag{2.22a}$$

Equation (2.22) represents the most general form of the delta-rule for the correction of self-learning procedures; it will be intensively used in Chapter 8 to describe the back-propagation of errors algorithm.

In general, a good binary decision vector $\boldsymbol{W}$ for any sub-decision requires a lengthy iterative procedure. Such a procedure starts with an arbitrary $\boldsymbol{W}^{(0)}$ (which can contain random numbers) and a set of objects $\{\boldsymbol{X}_s\}$ all identically represented as m-dimensional objects for which the decisions that should be made are already known. Using this knowledge we can then decide whether the correction of a currently active decision vector $\boldsymbol{W}$ is necessary.

The learning starts with sequentially testing all objects $\boldsymbol{X}$ and correcting $\boldsymbol{W}$ whenever needed. After finishing one pass over the entire set of objects, the decision vector $\boldsymbol{W}$ has changed considerably; hence, the objects previously classified correctly might be misclassified in the second pass. Therefore, the testing and correcting of the decision vector should be repeated as long as there is at least one error.

The results obtained by the linear learning machine method can be used in very complex decision processes. Jurs and Isenhour have used this method to predict molecular formulas from mass spectra, using a decision tree composed of 26 binary decisions. The scheme of the tree was even used as the cover design for the August 1971 issue of the journal *Analytical Chemistry* (Figure 2-10).

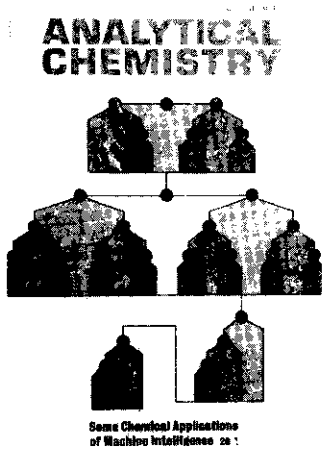

Figure 2-10: Cover design of the August 1971 issue of Analytical Chemistry, showing a decision tree with 26 decisions for predicting a molecular formula from the mass spectrum.

In spite of some success, the complex decision schemes obtained by linear learning machines are generally not very satisfactory, especially for large real-world problems. Therefore, after some initial enthusiasm, serious criticisms of this method were published (see Minsky and Papert); today, linear learning machines are mainly used with very restricted sets of data, and for education.

2.4 Transfer Functions in Neurons

The present model of a neuron consists of two distinct steps in obtaining output from the incoming signals. The first step, evaluation of the net input *Net*, was explained in the previous paragraph. We will

now have a closer look at the second step, in which a nonlinear transformation of the net input signal *Net* takes place.

We might come up with a model involving only a direct transformation of the input signals to the output; However, such a function would involve many input signals and many weights, and would at the same time be nonlinear. Therefore, instead of one complex transformation, which would be hard to justify and computationally difficult, a two-step procedure was introduced. In addition, these two steps seem more plausible in light of what we know about the functioning of biological neurons.

It would not be very convincing to represent the output of an artificial neuron as the weighted sum of the input signals, since the resulting signal *Net* can be a) very large and b) negative. The latter seems particularly unrealistic. After all, the neuron either "fires" or it does not; even though the firing frequency differs from stimulus to stimulus, the idea of a negative firing frequency is just not reasonable.

We are forced, then, to make the net input to the neuron (Equation 2.10) undergo an additional, nonlinear transformation:

$$out = f(Net) \tag{2.23}$$

called a *transfer function.*

What can we do to the artificial neuron's output signal *out* in order to make it more realistic? Because of the physical limitations on the size and frequency of brain signals, these conditions are quite simple: first, the final output signal of a neuron should be non-negative, whatever its magnitude; and second, it should be continuous and confined to a specified interval, say between zero and one. (In many publications, such a transfer function is called a **squashing** function because it squashes the output into a small interval.)

There are several functions satisfying the above conditions, but we will take a closer look at only three of them.

Hard-limiter: The first transfer function we will look at is, as Figure 2-11 shows, very simple indeed. It is called a *hard-limiter*, hl for short. It can have only two values: zero or one. There is one important point for this function, called the threshold value, ϑ. The value output by the hard-limiter depends on where the threshold value is set; thus, the threshold parameter ϑ decides whether the neuron will fire or not.

If the input value *Net* is $\geq \vartheta$, the output *out* will be one; otherwise it will be zero. Mathematically, this function (the *binary hard-limiter*) can be written:

$$out = \mathrm{hl}(Net, \vartheta) = \begin{cases} 1 & \text{if } Net \geq \vartheta \\ 0 & \text{if } Net < \vartheta \end{cases} \tag{2.24}$$

or as an expression that lends itself to easy programming:

$$out = \mathrm{hl}(Net, \vartheta) = 0.5 \ \mathrm{sign}(Net - \vartheta) + 0.5 \tag{2.24a}$$

where the function sign(·) has only two values, +1 and −1, for positive and negative arguments, respectively. If the threshold value $\vartheta = 0$, then the hard-limiter (2.24) assigns 0 to all negative values of *Net* and +1 to all positive ones. Since the hard-limiter is very convenient for giving straight answers (yes or no), it is often used for final outputs where definite answers are required.

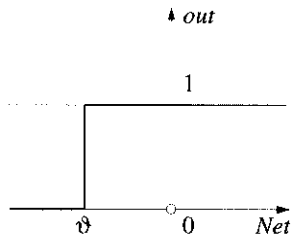

Figure 2-11: A binary hard limiter hl(*Net*,ϑ). If ϑ is set to less than zero the limiter jumps to a value of 1 on the negative side of the argument *Net*, if ϑ is larger than zero the transition to 1 occurs on the positive side.

The hard-limiter giving 0 and 1 as output is not very suitable for many applications. Therefore, a slightly modified form of (2.24) giving +1 and −1 instead of 0 and 1 may be used for output. This *bipolar hard-limiter* can be written:

$$out = \mathrm{hl}(Net, \vartheta) = \begin{cases} 1 & \text{if } Net \geq \vartheta \\ -1 & \text{if } Net < \vartheta \end{cases} \tag{2.25}$$

or:

$$out = \mathrm{hl}(Net, \vartheta) = \mathrm{sign}(Net - \vartheta)$$

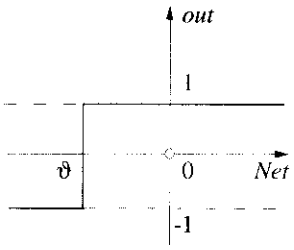

Figure 2-12: A bipolar hard-limiter.

The picture of the bipolar hard-limiter (Figure 2-12) is very similar to the binary hard-limiter (Figure 2-11).

Threshold logic: The second form of the transfer function we will discuss is the so-called *threshold logic*, tl, shown in Figure 2-13. In some respects, it is similar to the hard-limiter but has, in addition, a *swap interval*, within which *out* is linearly proportional to *Net*. The width of this interval is determined by a parameter α; the interval starts at ϑ and has a width of 1/α.

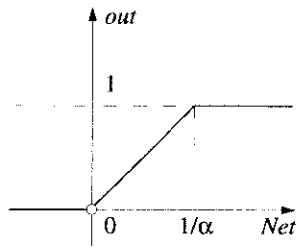

Figure 2-13: Threshold logic.

The calculation of the threshold logic function is very simple because the functions max(*a*, *b*) and min(*a*, *b*), which determine the maximum or minimum of two (or more) values between the two parameters *a* and *b* are easy to calculate.

Regardless of the value of the parameter *a*, max(0, *a*) is always **larger** than or equal to zero and min(1, *a*) is always less than or equal

to 1. Setting the value of a equal to *Net* and combining both functions, the following expression is obtained:

$$y = \max(0, \min(1, Net)) \tag{2.26}$$

$y(Net)$ is called a *threshold logic* (Figure 2-13).

By substituting the more general expression $\alpha(Net - \vartheta)$ for *Net*, the threshold logic is obtained in a form that can be used as a transfer function in neurons (Figure 2-14):

$$\begin{aligned} out &= \mathrm{tl}(Net, \alpha, \vartheta) = \\ &= \max\{0, \min[1, \alpha(Net - \vartheta)]\} \end{aligned} \tag{2.27}$$

α is called the *reciprocal width* of the swap interval.

Table 2-1 shows how the threshold logic behaves for two different sets of values of α and ϑ. The shaded area is the interval of values of *Net* in which the tl(*Net*, α, ϑ) function response is linear. It can be seen that the size of the linear response interval is inversely proportional to the parameter ($\alpha\vartheta$); if positive, it shifts this interval towards negative *Net* values, and *vice versa*.

This means that the linear transfer interval of the tl function starts at ϑ. If one does not need the actual values of ϑ, Expression (2.27) can be rewritten in a slightly modified form by the substitution:

$$\alpha\vartheta = \vartheta' \tag{2.28}$$

This actually does not change anything; it just puts the variable expression into the more common form:

$$\mathrm{tl}(Net, \alpha, \vartheta') = \max[0, \min(1, \alpha Net - \vartheta')] \tag{2.29}$$

Both the hard-limiter and the threshold logic are very convenient for computer programs. The threshold logic, for example can be used where a linear output is desired over the entire output signal range. However, due to the fact that they may contain singularities, they are of little theoretical use.

Sigmoidal function: The most widely used transfer function in various neural network applications is the so called *sigmoidal function*, sf, shown in Figure 2-15. It is somewhat time-consuming for numerical calculation, especially if tens of thousands of neurons are involved, but nevertheless it is used so often that it is worthwhile to take a closer look at it:

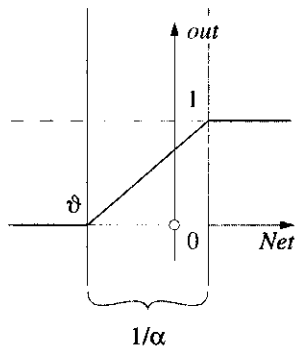

Figure 2-14: The threshold logic as a transfer function; the width of the linear response as a function of input is proportional to $1/\alpha$. The beginning of the swap interval is located at ϑ.

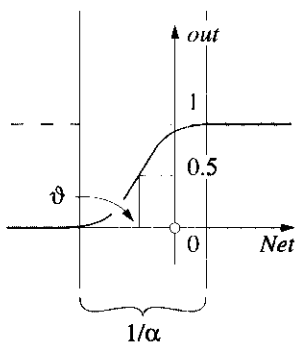

Figure 2-15: Sigmoidal transfer function.

Net	*v*	*out*	*Net*	$\alpha(Net-\vartheta)$ α=0.333, ϑ = –1	*v*	*out*	*Net*	$\alpha Net-\vartheta'$ α=0.333, ϑ' = –1	*v*	*out*
–10.0	–10.0	0.0	–10.0	–3.00	–3.00	0.000	–10.0	–2.333	–2.333	0.000
–5.0	–5.0	0.0	–5.0	–1.33	–1.33	0.000	–5.0	–0.667	–0.667	0.000
–3.0	–3.0	0.0	–3.0	–0.667	–0.667	0.000	**–3.0**	0.000	0.000	**0.000**
–2.8	–2.8	0.0	–2.8	–0.600	–0.600	0.000	–2.8	0.067	0.067	**0.067**
–2.6	–2.6	0.0	–2.6	–0.533	–0.533	0.000	–2.6	0.133	0.133	**0.133**
–2.4	–2.4	0.0	–2.4	–0.467	–0.467	0.000	–2.4	0.200	0.200	**0.200**
–2.2	–2.2	0.0	–2.2	–0.400	–0.400	0.000	–2.2	0.267	0.267	**0.267**
–2.0	–2.0	0.0	–2.0	–0.333	–0.333	0.000	–2.0	0.333	0.333	**0.333**
–1.8	–1.8	0.0	–1.8	–0.267	–0.267	0.000	–1.8	0.400	0.400	**0.400**
–1.6	–1.6	0.0	–1.6	–0.200	–0.200	0.000	–1.6	0.467	0.467	**0.467**
–1.4	–1.4	0.0	–1.4	–0.133	–0.133	0.000	–1.4	0.533	0.533	**0.533**
–1.2	–1.2	0.0	–1.2	–0.067	–0.067	0.000	–1.2	0.600	0.600	**0.600**
–1.0	–1.0	0.0	**–1.0**	0.000	0.000	**0.000**	–1.0	0.667	0.667	**0.667**
–0.8	–0.8	0.0	–0.8	0.067	0.067	**0.067**	–0.8	0.733	0.733	**0.733**
–0.6	–0.6	0.0	–0.6	0.133	0.133	**0.133**	–0.6	0.800	0.800	**0.800**
–0.4	–0.4	0.0	–0.4	0.200	0.200	**0.200**	–0.4	0.867	0.867	**0.867**
–0.2	–0.2	0.0	–0.2	0.267	0.267	**0.267**	–0.2	0.933	0.933	**0.933**
0.0	0.0	**0.0**	0.0	0.333	0.333	**0.333**	**0.0**	1.000	1.000	**1.000**
0.2	0.2	**0.2**	0.2	0.400	0.400	**0.400**	0.2	1.067	1.000	1.000
0.4	0.4	**0.4**	0.4	0.467	0.467	**0.467**	0.4	1.133	1.000	1.000
0.6	0.6	**0.6**	0.6	0.533	0.533	**0.533**	0.6	1.200	1.000	1.000
0.8	0.8	**0.8**	0.8	0.600	0.600	**0.600**	0.8	1.267	1.000	1.000
1.0	1.0	**1.0**	1.0	0.667	0.667	**0.677**	1.0	1.333	1.000	1.000
1.2	1.0	1.0	1.2	0.733	0.733	**0.733**	1.2	1.400	1.000	1.000
1.4	1.0	1.0	1.4	0.800	0.800	**0.800**	1.4	1.467	1.000	1.000
1.6	1.0	1.0	1.6	0.867	0.867	**0.867**	1.6	1.533	1.000	1.000
1.8	1.0	1.0	1.8	0.933	0.933	**0.933**	1.8	1.600	1.000	1.000
2.0	1.0	1.0	**2.0**	1.000	1.000	**1.000**	2.0	1.667	1.000	1.000
5.0	1.0	1.0	5.0	2.000	1.000	1.000	5.0	2.667	1.000	1.000
10.0	1.0	1.0	10.0	3.667	1.000	1.000	10.0	3.333	1.000	1.000
	$v = \min(1, Net)$ or $v = \min\|1, \alpha(Net-\vartheta)\|$ *and*				$out = \max\{0, \min\|1, \alpha(Net-\vartheta)\|\}$			cf. Equation (2.27)		
					$out = \max[0, \min(1, \alpha Net-\vartheta')]$			cf. Equation (2.29)		

Table 2-1: The threshold logic function *tl*: effects of the parameters α and ϑ on the neuron's output *out* for different net input values *Net*. The left side of Table 2-1 (columns 1 through 3) shows the „normalised" *tl* function with α=1 and ϑ=0, therefore the swap of length 1 begins at *Net*=0 and becomes 1 at *Net*=1. To make the swap interval start at ϑ, use Equation (2.27): columns 4 to 7. Equation (2.29) is used in columns 8 to 11: the swap interval starts at ϑ'/α, i.e.: at $-1/0.333 = -3$ in our case. From the 7th and 11th columns it can be seen that the swap interval is equal to $1/\alpha$. For α=0.333, the swap is 3 units long extending from –1 to +2 or from –3 to 0, depending on ϑ or ϑ', respectively.

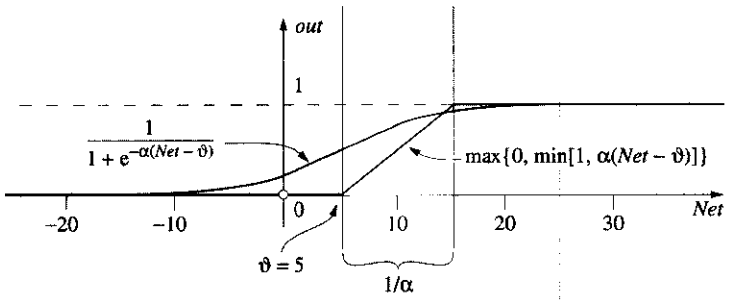

Figure 2-16: Comparison of the threshold logic and sigmoidal transfer functions if both have the same parameters ($\alpha = 0.1$, $\vartheta = 5$). Note that in the case of *tl* the swap starts at $Net=\vartheta$, while at exactly the same point the *sf* has its inflection.

$$sf(Net, \alpha, \vartheta) = 1/\{1+\exp[-\alpha(Net-\vartheta)]\} \quad (a)$$

or: (2.30)

$$sf(Net, \alpha, \vartheta') = 1/\{1+\exp[-(\alpha Net-\vartheta')]\} \quad (b)$$

Note the similarity between the pairs of Equations (2.27) / (2.30a), and Equations (2.29) / (2.30b).

Table 2-2 is calculated using the sigmoidal function (2.30a) and (2.30b) with the same input as Table 2-1. Because the same parameters α, ϑ and ϑ' equal to 0.333, –1, and –1, respectively, were used, the arguments $\alpha(Net - \vartheta)$ and $\alpha Net - \vartheta'$ are the same in both Tables. Thus, the two transfer functions can be easily compared.

Obviously, with the **same pair of parameters** and the **same** *Net* values, the two equations have quite similar behavior; however, they give the response in slightly different regions. Figure 2-16 compares the behavior of the threshold logic and the sigmoidal function with exactly the same parameters.

Beginners often believe that the transfer function (either Equation (2.29) or (2.30)) is adjusted optimally if all net input signals fall into the quasilinear region shown in Figure 2-13 and in Table 2-2; this interpretation is completely false.

> The fundamental assumption of modern neural network theory is that the transfer signals are not linearly dependent on the net input.

Net $\alpha = 1.0$, $\vartheta = 0.0$	*out* Equat. (2.30a)	*out* Tab. 2-1 col. 3	*Net*	$\alpha(Net - \vartheta)$ α=0.333, $\vartheta = -1$	*out* Equat. (2.30a)	*out* Tab. 2-1 col. 7	*Net*	$\alpha Net - \vartheta'$ α=0.333, $\vartheta' = -1$	*out* Equat. (2.30b)	*out* Tab. 2-1 col. 11
−10.0	**0.000**	0.0	−10.0	−3.000	**0.047**	0.000	−10.0	−2.333	**0.088**	0.000
−5.0	**0.001**	0.0	−5.0	−1.333	**0.209**	0.000	−5.0	−0.667	**0.339**	0.000
−3.0	**0.047**	0.0	−3.0	−0.667	**0.339**	0.000	**−3.0**	0.000	**0.500**	**0.000**
−2.8	**0.057**	0.0	−2.8	−0.600	**0.354**	0.000	−2.8	0.067	**0.517**	**0.067**
−2.6	**0.069**	0.0	−2.6	−0.533	**0.370**	0.000	−2.6	0.133	**0.133**	**0.133**
−2.4	**0.083**	0.0	−2.4	−0.467	**0.385**	0.000	−2.4	0.200	**0.550**	**0.200**
−2.2	**0.100**	0.0	−2.2	−0.400	**0.401**	0.000	−2.2	0.267	**0.566**	**0.267**
−2.0	**0.119**	0.0	−2.0	−0.333	**0.418**	0.000	−2.0	0.333	**0.562**	**0.333**
−1.8	**0.142**	0.0	−1.8	−0.267	**0.434**	0.000	−1.8	0.400	**0.599**	**0.400**
−1.6	**0.168**	0.0	−1.6	−0.200	**0.450**	0.000	−1.6	0.467	**0.615**	**0.467**
−1.4	**0.198**	0.0	−1.4	−0.133	**0.467**	0.000	−1.4	0.533	**0.630**	**0.533**
−1.2	**0.231**	0.0	−1.2	−0.067	**0.483**	0.000	−1.2	0.600	**0.646**	**0.600**
−1.0	**0.269**	0.0	**−1.0**	0.000	**0.500**	**0.000**	−1.0	0.667	**0.661**	**0.667**
−0.8	**0.310**	0.0	−0.8	0.067	**0.517**	**0.067**	−0.8	0.733	**0.675**	**0.733**
−0.6	**0.354**	0.0	−0.6	0.133	**0.533**	**0.133**	−0.6	0.800	**0.690**	**0.800**
−0.4	**0.401**	0.0	−0.4	0.200	**0.550**	**0.200**	−0.4	0.867	**0.704**	**0.867**
−0.2	**0.450**	0.0	−0.2	0.267	**0.566**	**0.267**	−0.2	0.933	**0.718**	**0.933**
0.0	**0.500**	**0.0**	0.0	0.333	**0.562**	**0.333**	**0.0**	1.000	**0.731**	**1.000**
0.2	**0.550**	**0.2**	0.2	0.400	**0.599**	**0.400**	0.2	1.067	**0.744**	1.000
0.4	**0.599**	**0.4**	0.4	0.467	**0.615**	**0.467**	0.4	1.133	**0.756**	1.000
0.6	**0.646**	**0.6**	0.6	0.533	**0.630**	**0.533**	0.6	1.200	**0.768**	1.000
0.8	**0.690**	**0.8**	0.8	0.600	**0.646**	**0.600**	0.8	1.267	**0.780**	1.000
1.0	**0.731**	**1.0**	1.0	0.667	**0.661**	**0.677**	1.0	1.333	**0.791**	1.000
1.2	**0.768**	1.0	1.2	0.733	**0.675**	**0.733**	1.2	1.400	**0.802**	1.000
1.4	**0.802**	1.0	1.4	0.800	**0.690**	**0.800**	1.4	1.467	**0.813**	1.000
1.6	**0.832**	1.0	1.6	0.867	**0.704**	**0.867**	1.6	1.533	**0.822**	1.000
1.8	**0.858**	1.0	1.8	0.933	**0.718**	**0.933**	1.8	1.600	**0.832**	1.000
2.0	**0.881**	1.0	**2.0**	1.000	**0.731**	**1.000**	2.0	1.667	**0.841**	1.000
5.0	**0.993**	1.0	5.0	2.000	**0.881**	1.000	5.0	2.667	**0.935**	1.000
10.0	**1.000**	1.0	10.0	3.667	**0.975**	1.000	10.0	4.333	**0.987**	1.000
			sf (*Net*, α, ϑ) $= 1 / \{1+\exp[-\alpha(Net - \vartheta)]\}$				cf. Equation (2.30a)			
			sf (*Net*, α, ϑ') $= 1 / \{1+\exp[-(\alpha Net - \vartheta')]\}$				cf. Equation (2.30b)			

Table 2-2: Comparison of the sigmoidal function output, *sf*, with the threshold logic function, *tl*: effects of the parameters α and ϑ on the neuron's output *out* for different net input values, *Net*. The swap interval, i.e., the transition between output of zero and one, is much broader for the sigmoidal function compared to that of the threshold logic. It can be clearly seen that the inflection of the *sf*, *out* = 0.5, occurs always at the same position as the beginning of the swap of the corresponding threshold logic function *tl*.

Of course, some neurons will show a linear relation between *Net* and *out*; however, it is the nonlinearity of the transfer function that makes the neural network so flexible for adjusting to different learning situations. A linear relation between the net input and its output may be desirable for some special cases, but nonlinearity should usually prevail in a neural network.

In Figure 2-17, the input is assumed to be confined to the range from *A* to *B*. In the topmost case the sigmoidal transfer function does not produce an output at all (an output of zero means that the neuron does not fire). In the second, the sigmoidal transfer function produces a maximum output signal no matter what the net input is. In the third, the response is almost linearly proportional to the input. In the fourth case, signals between *B* and *B'* all produce maximum output, while inputs between *A* and *B'* make full use of the transition interval of the sigmoidal transfer function. And in the last example, all inputs between *A* and *A'* produce no output; only signals between *A'* and *B* give a nonzero output.

Even those neurons whose transfer functions preclude firing at all during the entire training period (Figure 2-17, top) can be of interest in some applications; it is thought that their function is merely to wait for some event not anticipated in the training phase that might stimulate a response.

Instead of the mentioned *Net* input (Equation (2.1)) and one of the above transfer functions, more complex functions can easily be found. However, we are looking for the simplest adequate description of a neuron.

We previously mentioned the difficulties involving the derivatives of the hard-limiter and the threshold transfer functions because they contain singularities. Now, let's have a look at the derivative of the sigmoidal function (2.30), which will be used later (Figure 2-18). For clarity we will write Equation (2.30) as:

$$\mathrm{sf}(x) = 1/[1+\exp(-x)] \tag{2.31}$$

The derivative is obtained according to the rule for quotients. In case your calculus is a bit rusty, we will carry out the derivation in detail:

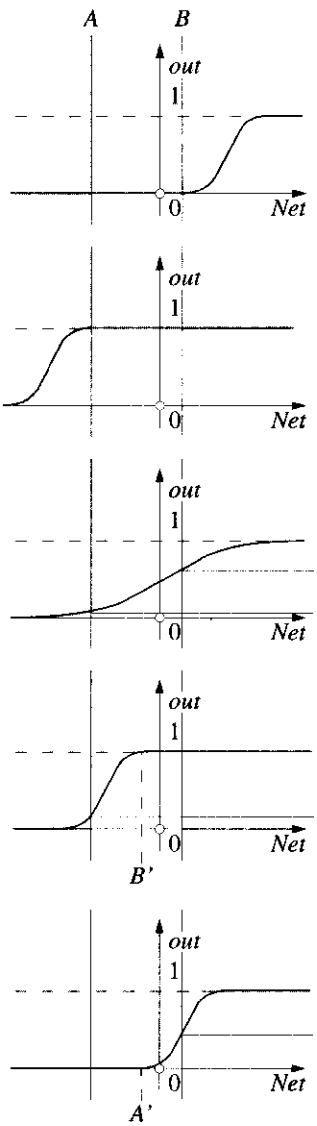

Figure 2-17: Different adaptations of neural sigmoidal functions for the same range of signals; the inputs *Net* are all confined to the interval (*A*, *B*).

$$
\begin{aligned}
d(sf(x))/dx &= -1/[1+\exp(-x)]^2 \{ d[1+\exp(-x)]/dx \} = \\
&= \exp(-x)/[1+\exp(-x)]^2 = \\
&= sf(x)\exp(-x)/[1+\exp(-x)] = \\
&= sf(x)\{[-1+1+\exp(-x)]/[1+\exp(-x)]\} = \\
&= sf(x)[-sf(x)+1] = \\
&= sf(x)[1-sf(x)] \qquad (2.32)
\end{aligned}
$$

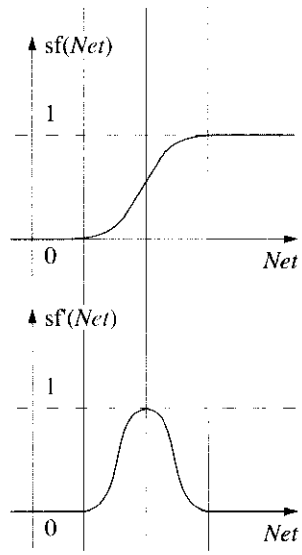

Figure 2-18: The derivative of the sigmoidal transfer function.

> The above equation clearly shows that in the flat regions of sf(x) (where sf(x) = 0 or sf(x) = 1), the derivative is zero. This fact will turn out to be very important later, when we investigate when and where neural networks learn best.

The swap interval between two states, exhibited by two of the three transfer functions (the threshold logic and the sigmoidal functions) is in sharp contrast to the hard-limiter's *yes/no*- (*true/false*-) logic. The threshold and sigmoidal functions, (2.29) and (2.30), are actually not logic elements at all. They may hold all states or values between the two extreme assertions *yes* and *no*, or *true* and *false*; this possibility forms the link between artificial neural networks and "fuzzy" logic, a field that has recently gained a lot of attention in artificial intelligence and neural network research.

By changing the parameters α and ϑ, the swap interval (the "fuzziness") of the decisions can be influenced. If the extent of fuzziness can be influenced through the choice of learning procedure, the results (predictions) obtained can be quantitatively evaluated; the prediction abilities of such procedures would be much more "human-like" than when complex decisions are based on hierarchical decision trees in which each individual decision has to be made in a logical (*yes/no*) manner. The existence of the swap interval and "fuzzy" logic in the transfer function is one of the greatest assets of this model.

2.5 Bias

The addition of an extra parameter, called bias, to the decision function increases its adaptability to the decision problem it is designed to solve. We will illustrate the importance of the bias through the linear learning machine. However, the conclusions are valid for many other neural network models, in particular, for the back-propagation algorithm (see Chapter 8). The example at the top of Figure 2-19 shows that the family of straight line functions $y = \alpha x$ cannot separate the class of points A, B and C from the class represented by point D. On the other hand, it is easy to separate the class (A, B, C) from the class (D) by introducing a constant ϑ, the bias, to the straight line decision function $y = \alpha x + \vartheta$, as shown in the lower part of Figure 2-19.

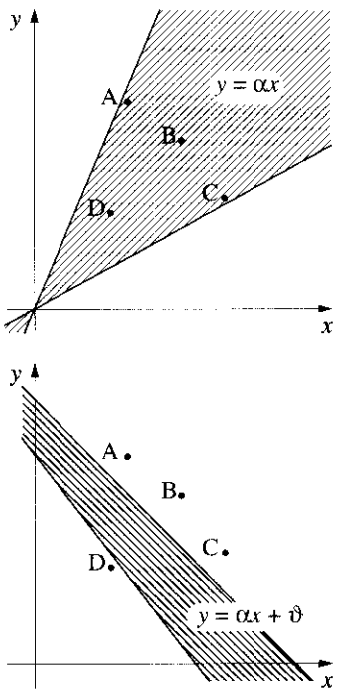

Figure 2-19: Introducing the bias, ϑ, to the decision function separating D from the points A, B and C.

We have seen that in order to describe an artificial neuron we must know two types of parameters: the set of weights and the parameters of the transfer function. There are as many weights as there are signals entering the neuron. Generally, they are initialized as small random numbers. The actual interval within which these random weights are selected roughly depends on the number of weights in the neuron; a fair choice in applications where the normalization of weights is not strictly required is to set the interval so that the squared sum of the weights in one neuron is about 0.5.

Now the problem is how to treat the parameters of the transfer function. If we leave aside the swap interval of the transfer function for a moment, the crucial point in all three transfer functions is the threshold ϑ, i.e. the point where the neuron starts to react.

> In this paragraph we will show that all parameters determining the artificial neuron (weights, interval, and threshold) can all be formally treated in exactly the same way.

Let us review the two main equations that describe the functioning of an artificial neuron:

$$Net = w_1x_1 + w_2x_2 + \ldots + w_ix_i + \ldots + w_mx_m = \boldsymbol{WX} \qquad (2.7)$$

$$\mathrm{sf}(Net, \alpha, \vartheta) = 1 / \{ 1 + \exp[-\alpha (Net - \vartheta)] \} \qquad (2.30a)$$

where:

- w_i is the synaptic strength (the weight) of the synapse i,
- x_i is the signal coming from a connected neuron entering our neuron at the synapse i,
- m is number of synapses in our neuron, i.e., the number of input signals,
- *Net* is the net input (accumulation of all inputs) formed within our neuron,
- α is the reciprocal width of the swap interval (see Equation (2.27)), and
- ϑ is the threshold value of the sigmoidal transfer function sf (see Equation (2.24a)).

The transfer function is so simple that we need to consider only its argument, $\text{arg} = \alpha Net - \vartheta$, to show our conjecture:

$$\text{arg} = \alpha w_1 x_1 + \alpha w_2 x_2 + \ldots + \alpha w_m x_m - \alpha\vartheta \tag{2.33}$$

Because all parameters on the right hand side of Equation (2.33) are unknown (although it is assumed that the signals x_i will be known after the learning period), we may substitute the products of two unknown values αw_i by the single value w_i', and $-\alpha\vartheta$ by ϑ'. Equation (2.33) now becomes:

$$\text{arg} = w'_1 x_1 + w'_2 x_2 + \ldots + w'_m x_m + \vartheta' \tag{2.34}$$

Now, if we regard the scalar value ϑ' as a product of ϑ and a component x_{m+1} which is **always equal to 1**, we get the following summation:

$$\text{arg} = w'_1 x_1 + w'_2 x_2 + \ldots + w'_m x_m + \vartheta' x_{m+1} \tag{2.35}$$

If we label ϑ' as w'_{m+1}, we actually have created a product of w'_{m+1} and a signal x_{m+1} (always equal to 1); since this is completely analogous to the other products in the series, we can extend the summation by one more element:

$$\begin{aligned} \text{arg} &= w'_1 x_1 + w'_2 x_2 + \ldots + w'_m x_m + w'_{m+1} x_{m+1} = \\ &= \sum_{i=1}^{m+1} w'_i x_i \end{aligned} \tag{2.36}$$

Now, by inserting the evaluated arg (2.36) back into the sigmoidal transfer function (2.30), and dropping the unnecessary "prime" on the w's, we obtain:

$$\mathrm{sf}\,(Net, \alpha, \vartheta) = 1/\left\{1 + \exp\left(-\sum_{i=1}^{m+1} w_i x_i\right)\right\} \tag{2.37}$$

It is evident that the real output produced by the neuron, as described by the sigmoidal function sf(Net, α, ϑ), depends only on the (m+1)-dimensional weight vector $\boldsymbol{W}$ and the (m+1)-dimensional signal $\boldsymbol{X}$:

$$\begin{aligned} \boldsymbol{W} &= (w_1, w_2, \ldots, w_m, w_{m+1}) \\ \text{and} & \\ \boldsymbol{X} &= (x_1, x_2, \ldots, x_m, 1) \end{aligned} \tag{2.38}$$

The extra weight which should be present in artificial neurons of this design (and which always receives an input value of 1) is the bias.

> We can regard the threshold value ϑ and the constant α simply as an additional "synaptic strength" called bias to which a signal of value 1 is always transmitted.

We will now explore the influence of the bias on a modification of the example considered in Figure 2-8. Using $\boldsymbol{W}_1$ (the decision vector at the root of the decision tree in Figure 2-8) it is possible to decide whether a point $\boldsymbol{X}(x_1, x_2)$ is above or below the abscissa, according to the sign of the dot product $\boldsymbol{W}_1\boldsymbol{X}$:

if $\boldsymbol{W}_1\boldsymbol{X} \geq 0$ $\quad$ $\boldsymbol{X}$ is above or on the abscissa

if $\boldsymbol{W}_1\boldsymbol{X} < 0$ $\quad$ $\boldsymbol{X}$ is below the abscissa

At this point we will explain how the decision vector $\boldsymbol{W}_1$ is obtained, using a slightly more complicated example. Let us assume that we have four points $\boldsymbol{X}_1 = (3, 2)$, $\boldsymbol{X}_2 = (-1, -1)$, $\boldsymbol{X}_3 = (1, -3)$ and $\boldsymbol{X}_4 = (4, 1)$, of which the first two belong to the category C_1 and the other two to C_2. Plotting these four points into the xy-plane (Figure 2-20),

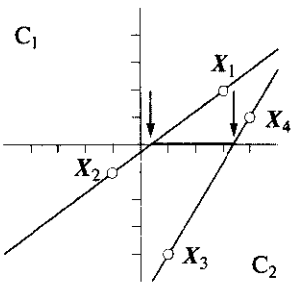

Figure 2-20: Four points in the xy-plane belonging to two different categories, $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ belong to C_1, while $\boldsymbol{X}_3$ and $\boldsymbol{X}_4$ belong to C_2. The separating line must be somewhere between the two lines indicated.

we realize that it should not be difficult to find a decision line separating these four points; it could be any line within the white area.

The solution could be obtained by writing down four trivial logical statements:

$$\begin{aligned}
&\text{if } (\boldsymbol{X} = \boldsymbol{X}_1), \quad \text{then } (\boldsymbol{X} \text{ is in } C_1)\\
&\text{if } (\boldsymbol{X} = \boldsymbol{X}_2), \quad \text{then } (\boldsymbol{X} \text{ is in } C_1)\\
&\text{if } (\boldsymbol{X} = \boldsymbol{X}_3), \quad \text{then } (\boldsymbol{X} \text{ is in } C_2)\\
&\text{if } (\boldsymbol{X} = \boldsymbol{X}_4), \quad \text{then } (\boldsymbol{X} \text{ is in } C_2)
\end{aligned}$$

This is not a generally applicable procedure: it may not tell us anything about other points in the plane. In other words, the above procedure is **not a model** that can be used for predictions.

Using analytical geometry, we can write an equation for a straight line separating these four points. This line **is** a model, because it can be used as a decision line for **any** point in the plane. However, we do not favor the way it is obtained because, in a general case, complex handling of multidimensional planes and lines is required.

Instead, we would like to obtain the result simply by learning from a series of points (learning-by-example), and have it apply to any point, not just these four.

That is, we would like to obtain a decision vector $\boldsymbol{W}$ which will give a positive dot product $\boldsymbol{W}\boldsymbol{X}_i$ for points $\boldsymbol{X}_i$ belonging to C_1, and negative for points belonging to C_2. No matter how hard we try, we cannot obtain a solution with a 2-dimensional $\boldsymbol{W}$. However, Equation (2.38) shows that flexibility of adaptation can be obtained by augmenting the weight vector $\boldsymbol{W}$ by one additional weight w_{m+1}. An appropriate weight vector should therefore have three dimensions.

The only remaining question is: what weight vector should we start with? The simplest guess is the vector (1, 1, 1).

Hence, we start with:

$$\boldsymbol{X}_1 = (3,\ 2,\ 1), \quad \boldsymbol{X}_2 = (-1, -1, 1), \text{ category 1, dot product} \geq 0$$

$$\boldsymbol{X}_3 = (1, -3, 1), \quad \boldsymbol{X}_4 = (\ 4,\ 1, 1), \text{ category 2, dot product} < 0$$

and the starting weight vector $\mathrm{W}^{(0)}$:

$$\boldsymbol{W}^{(0)} = (1, 1, 1)$$

For correcting $\boldsymbol{W}$ we will use the delta-rule as introduced by Equation (2.12), $\Delta\boldsymbol{W} \sim \delta\boldsymbol{X}$, through Equation (2.20)

$$\Delta\boldsymbol{W} = \boldsymbol{W}^{(new)} - \boldsymbol{W}^{(old)} = -\left(2\eta \boldsymbol{W}^{(old)}\boldsymbol{X}/\|\boldsymbol{X}\|^2\right)\boldsymbol{X} \quad (2.20)$$

which is clearly identical to (2.12). Because the proportional coefficient δ is written as:

$$\delta = -\left(2\eta \boldsymbol{W}^{(old)} \boldsymbol{X} / \|\boldsymbol{X}\|^2 \right) \tag{2.39}$$

for each new vector $\boldsymbol{X}$, $\Delta\boldsymbol{W}$ can be calculated using equation (2.20) as shown in Table 2-3.

It is evident that the first product, $\boldsymbol{W}\boldsymbol{X}_1$, will yield a positive result, so no correction is necessary. The next product $\boldsymbol{W}\boldsymbol{X}_2$, which gives a **negative** dot product, requires a correction of $\boldsymbol{W}$ because $\boldsymbol{X}_2$ belongs to category C_1 (for simplicity we set η equal to 1). After this correction has been made, the new $\boldsymbol{W}$ is multiplied by the vector $\boldsymbol{X}_3$; since the wrong answer is produced by the multiplication, the correction is made again. This training continues until the correct prediction is achieved for all four points; a record of this is given in Table 2-3.

i	category of $\boldsymbol{X}_i$	$\boldsymbol{X}_i$	$\boldsymbol{W}$	$\boldsymbol{W}\boldsymbol{X}_i$	sign of $\boldsymbol{W}\boldsymbol{X}_i$	predicted category	result	$\|X_i\|^2$ (2.15a)	δ	$\delta\boldsymbol{X}_i$	new $\boldsymbol{W} = \boldsymbol{W} - \delta\boldsymbol{X}_i$
1	C_1	(3, 2, 1) *	(1.00, 1.00, 1.00) =	6.00	+	C_1	OK				
2	C_1	(−1, −1, 1) *	(1.00, 1.00, 1.00) =	−1.00	−	C_2	wrong	3.00	−0.67	(0.67, 0.67, −0.67)	(0.33, 0.33, 1.67)
3	C_2	(1, −3, 1) *	(0.33, 0.33, 1.67) =	1.00	+	C_1	wrong	11.00	0.18	(0.18, −0.55, 0.18)	(0.15, −0.88, 1.4
4	C_2	(4, 1, 1) *	(0.15, 0.88, 1.49) =	2.97	+	C_1	wrong	18.00	0.33	(1.33, 0.33, 0.33)	(−1.17, 0.55, 1.16
					end of the first cycle: 3 errors						
1	C_1	(3, 2, 1) *	(−1.17, 0.55, 1.16) =	−1.25	−	C_2	wrong	14.00	−0.18	(−0.54, −0.36, −0.18)	(−0.63, 0.91, 1.33)
2	C_1	(−1, −1, 1) *	(−0.63, 0.91, 1.33) =	1.06	+	C_1	OK				
3	C_2	(1, −3, 1) *	(−0.63, 0.91, 1.33) =	−2.02	−	C_2	OK				
4	C_2	(4, 1, 1) *	(−0.63, 0.91, 1.33) =	−0.29	−	C_2	OK				
					end of the second cycle: 1 error						
1	C_1	(3, 2, 1) *	(−0.63, 0.91, 1.33) =	1.25	+	C_1	OK				
2	C_1	(−1, −1, 1) *	(−0.63, 0.91, 1.33) =	1.06	+	C_1	OK				
3	C_2	(1, −3, 1) *	(−0.63, 0.91, 1.33) =	−2.02	−	C_2	OK				
4	C_2	(4, 1, 1) *	(−0.63, 0.91, 1.33) =	−0.29	−	C_2	OK				
					end of the third cycle: 0 error						
		final $\boldsymbol{W}$	(−0.63, 0.91, 1.33)								

Table 2-3: Training events in the process of adaptation of the weight vector to classify four objects correctly.

If the training had started with a different weight vector and/or a different sequence of training objects, the resulting weight vector

would be slightly different, although it would correctly predict all four points.

As mentioned before, by adding the bias we have moved our problem from the two-dimensional space, where the solution should be a line, to a three-dimensional space, where the solution is a plane. The solution is defined by a vector $\boldsymbol{W}^{(final)} = (-0.63, 0.91, 1.33)$ that is perpendicular to the decision plane (see Table 2-3).

The sequence of changes of the weight vector $\boldsymbol{W}$ shown in Table 2-3 is called learning and the entire procedure, a *linear learning machine*. It is tedious to carry out even this very simple example by hand, let alone cases with hundreds of points and tens of decisions. Therefore, the reader is encouraged to do a little programming.

> A solution may be obtained in all linearly separable cases by augmenting the decision vector $\boldsymbol{W}$ with an additional component, the bias, and the vectors representing objects with an additional component equal to 1.

With different initial guesses of the weight vector $\boldsymbol{W}$, or different sequence orders of the four objects for learning, we will get different results. However, all of the final weight vectors $\boldsymbol{W}$ will be equally satisfactory for making a decision for any point. Not only $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3$, $\boldsymbol{X}_4$, but **all** the points above the line $(\boldsymbol{X}_1\boldsymbol{X}_2)$ or below the line $(\boldsymbol{X}_3\boldsymbol{X}_4)$ (see Figure 2-20) will be classified in the categories C_1 and C_2, respectively.

2.6 Graphical Representation of Artificial Neurons

Although there exist a number of suggestions for representing artificial neurons, none of them seems to be completely satisfactory. Figure 2-21 shows some historically ordered examples of how artificial neurons have been presented in the literature. Representing the neurons by rectangles (Kohonen) has the advantage that they show the connections of neurons in one layer, which makes it easy to understand how the neurons obtain the same multidimensional signal simultaneously. Additionally, it makes the vector representation of the

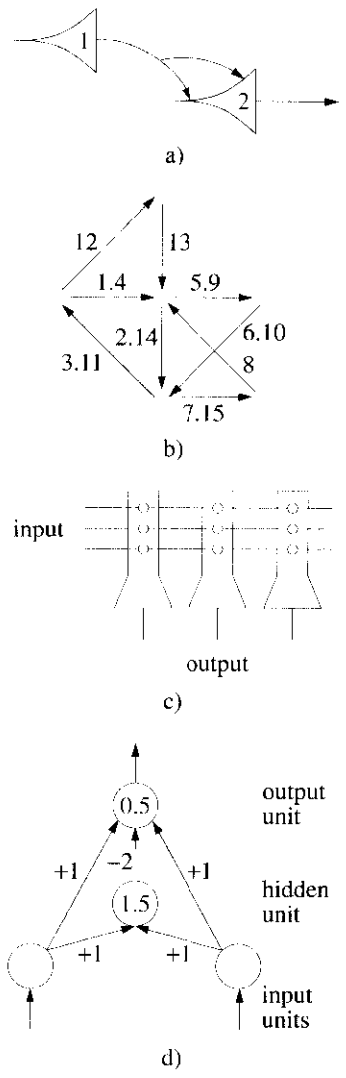

Figure 2-21: Representation of neurons by: a) McCulloch and Pitts, b) Hebb, c) Kohonen, d) Rumelhart.

neuron's weight vector ***W***, composed of one row or one column of weights, more visual.

In this book, a neuron will be represented by a circle divided by a horizontal line into two halves representing the summation and transfer functions. Figure 2-22 shows how the presentations of a neuron was developed in this chapter.

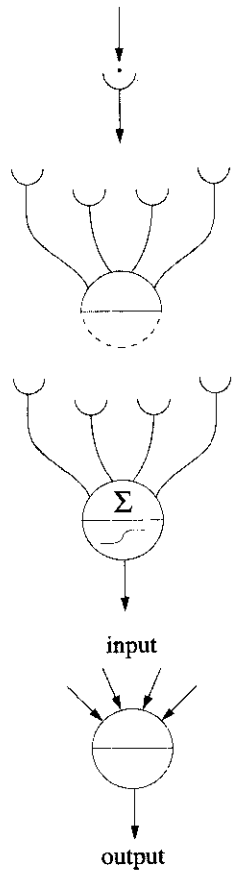

Figure 2-22: Development of the representation of a neuron in this chapter.

2.7 Essentials

- **net input:**

$$\begin{aligned} Net &= \boldsymbol{W}\boldsymbol{X} + \vartheta = \\ &= w_1 x_1 + w_2 x_2 + w_3 x_3 + \ldots + w_m x_m + \vartheta = \\ &= \sum_{i=1}^{m} w_i x_i + \vartheta \end{aligned} \tag{2.9}$$

- **transfer functions:**

hard-limiter (binary)

$$\mathrm{hl}(Net, \vartheta) = 0.5 \ \mathrm{sign}(Net - \vartheta) + 0.5 \tag{2.24}$$

hard-limiter (bipolar)

$$\mathrm{hl}(Net, \vartheta) = \mathrm{sign}(Net - \vartheta) \tag{2.25}$$

threshold logic

$$\mathrm{tl}(Net, \alpha, \vartheta) = \max\{0, \min[1, \alpha(Net - \vartheta)]\} \tag{2.27}$$

sigmoidal function

$$sf(Net, \alpha, \vartheta) = 1 / \left\{ 1 + \exp\left(- \sum_{i=1}^{m+1} w x_i \right) \right\} \tag{2.37}$$

- **learning:**

delta-rule

$$\Delta \boldsymbol{W} = \eta \, \delta \, \boldsymbol{X} \tag{2.22}$$

delta-rule for LLM

$$\boldsymbol{W}^{(new)} = \boldsymbol{W}^{(old)} - \left(2\eta \boldsymbol{W}^{(old)} \boldsymbol{X} / \|\boldsymbol{X}\|^2 \right) \boldsymbol{X} \tag{2.20}$$

2.8 References and Suggested Readings

2-1. S. Silbernagl and A. Despopoulos, *Color Atlas of Physiology*, Thieme, Stuttgart, FRG, 1991; *dtv-Atlas der Physiologie*, Thieme, Stuttgart, FRG, 1991.

2-2. S. W. Huffler and J. G. Nicholls, *From Neuron to Brain – A Cellular Approach to the Function of the Nervous System*, Sinauer Associates, Sunderland, MA, USA, 1986.

2-3. R. Forsyth and R. Rada, *Machine Learning: Applications in Expert Systems and Information Retrieval*, Ellis Horwood Ltd., Chichester, UK, 1986, Chapters 1 and 2.

2-4. N. J. Nilsson, *Learning Machines: Foundations of Trainable Pattern Classifying Systems*, McGraw-Hill, New York, USA, 1965, Chapter 3.

2-5. K. Varmuza, *Pattern Recognition in Chemistry*, Springer Verlag, Berlin, FRG, 1980.

2-6. S. Watanabe, *Pattern Recognition: Human and Mechanical*, Wiley, New York, USA, 1985.

2-7. J. Zupan, *Algorithms for Chemists*, John Wiley, Chichester, UK, 1989, Chapter 8.

2-8. M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, USA, 1969.

2-9. P. C. Jurs and T. L. Isenhour, "Some Chemical Applications of Machine Intelligence", *Anal. Chem.* **43** (1971) 20A – 36A.

2-10. D. O. Hebb, *The Organization of Behavior*, Wiley, New York, USA, 1949.

2-11. W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity", *Bull. Math. Biophys.* **5** (1943) 115 – 133.

2-12. T. Kohonen, *Self-Organization and Associative Memory*, Third Edition, Springer-Verlag, Berlin, FRG, 1989.

2-13. D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning Internal Representations by Error Propagation", in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Eds.: D. E. Rumelhart, J. L. McClelland, Vol. **1**, MIT Press, Cambridge, MA, USA, 1986, pp. 318 – 362.

2-14. J. Gasteiger and J. Zupan, "Neuronale Netze in der Chemie", *Angew. Chem.* **105** (1993) 510 – 536; "Neural Networks in Chemistry", *Angew. Chem. Int. Ed. Engl.* **32** (1993) 503 – 527.