3 Linking Neurons into Networks

learning objectives

- why networking is emphasized
- parallel processing
- organization of neurons into layers
- more about inputs and outputs
- architectures
- graphical representation of neural networks
- matrix notation of neural networks

3.1 General

In order to achieve realistic results, our model must conform to certain facts. For example, we know (see Smith-Churchland, Reference 3-7) that a neuron can fire again after approximately one millisecond, 10^{-3}s. Since the reaction time of most vertebrates is around one tenth of a second (10^{-1}s), we conclude that whatever happens in the brain to provoke a reaction must occur in less than 100 firing times. This is called the "hundred steps paradoxon".

The most striking implication of this is that the brain possesses a signal processing algorithm so powerful that it can handle the most difficult tasks we can imagine in only 100 steps. Because even the most powerful computers with their nanosecond (10^{-9}s) clock rates do not come close to such performance, we must attribute the brain's magnificent performance to something unique about its structure and functioning.

Since a single neuron approach, no matter how many weights such a neuron has, cannot find solutions to complicated real-world applications, this "unique something" must involve the way neurons are interconnected; we have come to think of the brain as a massively parallel processor. Hence, as we stated at the beginning of the book

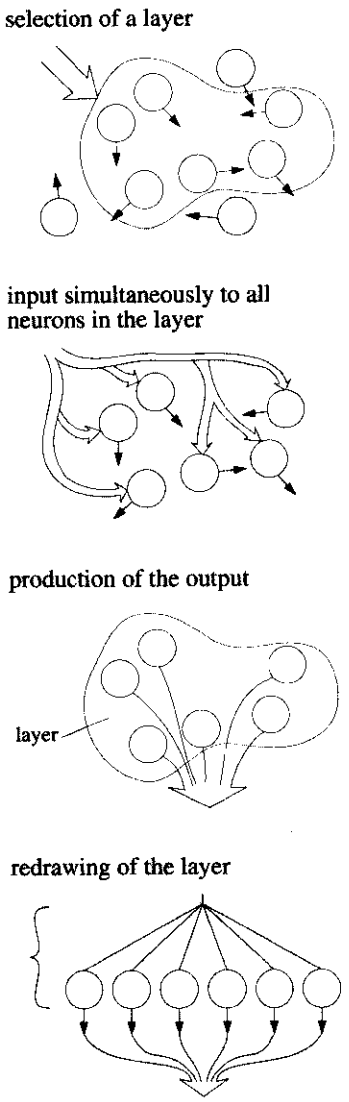

Figure 3-1: The steps in the formation of a neural network layer.

the emphasis in the phrase "neural networks" is clearly on the word "networks". In the rest of the book we will explore networks of neurons and their properties.

Our first exercise will be to organize a large number of neurons in such a way that **all** of them receive the same input $\boldsymbol{X}$ for processing at the same time (Figure 3-1, top). The production of a net input *Net* (Equation (2.7)) and the transformed output *out* (Equation (2.23)) then occurs in all neurons simultaneously (Figure 3-1, second and third part). As each neuron has a different set of weights, the otherwise identical procedure for generating output will produce as many different output signals as there are neurons. (A network "learns" by modifying its weights.)

Such a group of neurons producing a set of outputs simultaneously is called a layer (Figure 3-1 bottom). As each neuron j produces its own net input Net_j and output signal out_j, these individual signals of one layer can be combined to vectors, the net input signal vector, ***Net***, and the output vector, ***Out***. The output vector, ***Out***, can be used as an input vector, ***Inp*** or $\boldsymbol{X}$, to another layer of neurons.

The practical advantage of the neural network approach over conventional methods is that layers can be implemented on a parallel-operated computer chip.

Multilayer networks operate sequentially, i.e. the neurons in a layer do not receive the signals until the neurons from the previous ("upper") layer have produced them. Usually, no more than two or three layers of neurons are considered, and so the sequential link does not represent any substantial loss of time.

It should be noted, however, that until now neural network algorithms have generally been implemented on von-Neumann (i.e., sequential) computers; in such an implementation, the "simultaneous parallel" processing of a layer is actually performed **sequentially**. "Parallelism" is understood to mean that the neurons in one layer process information independently of each other. The output signals of one layer will be transmitted to the next layer only when **all** neurons of the first layer have finished their processing.

In this book, networks are drawn so that they "run" from top to bottom; other books may do the opposite (see Section 2.6).

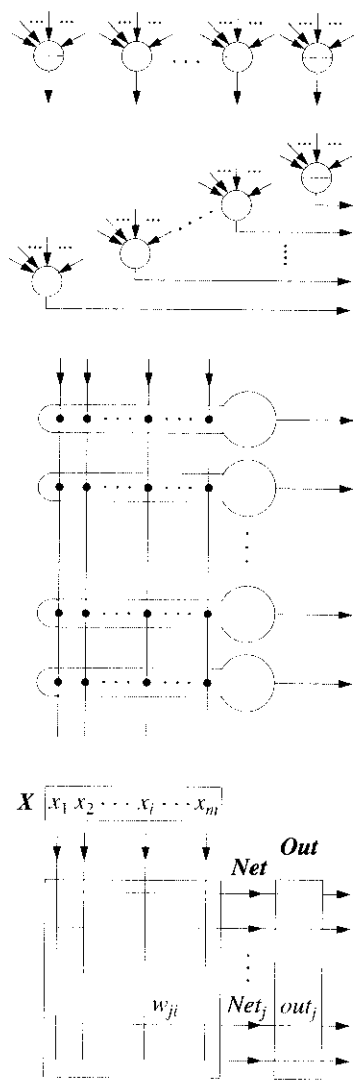

Figure 3-2: Four steps in the "evolution" from the biological to the matrix representation of neural networks.

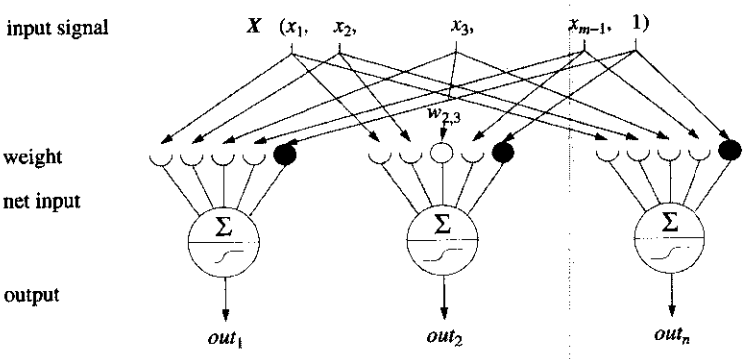

Figure 3-3: One-layer neural network.

3.2 One Layer

In our current model, a layer is a group of neurons **all** of which have the same number, m, of weights (synapses) and **all** receive the **same** m-dimensional input signal **simultaneously**. (What we have been calling a "layer" of neurons is often regarded as a "string", i.e., a linear arrangement of neurons.)

Neurons or layers of neurons are usually drawn to resemble biological neurons as well as possible: with circles acting as neural cells, and a number of interconnecting lines representing dendrites and axons. The synapses are placed somewhere along these lines. However, programmers and mathematicians prefer to think of the neural layer as a matrix of weights. Figure 3-2 shows a plausible way of obtaining a matrix notation from the "biological" one.

In the matrix of weights $\boldsymbol{W}$, the rows represent the neurons. Each row j can be labeled as a vector $\boldsymbol{W}_j$ representing a neuron j, consisting of m weights w_{ji}, $\boldsymbol{W}_j = (w_{j1}, w_{j2}, ..., w_{jm})$. All weights in the same column i, w_{ji} ($j = 1, 2, ..., n$), simultaneously obtain the same signal x_i. At a given moment, the entire input vector $\boldsymbol{X} = (x_1, x_2, ..., x_m)$ (which may come from an external source (sensor or instrument), or from another group of neurons) is input into the network, i.e. to the matrix $\boldsymbol{W}$. Since all weights w_{ji} in the entire matrix are simultaneously exposed to the corresponding input signals all products $w_{ji}\,x_i$ are made at the same time.

Figure 3-3 shows such a one-layer network composed of three neurons, each having the same number (five) of randomly generated weights. Each neuron in the layer obtains the same set of m signals (x_1, x_2, x_3, ..., x_{m-1}, 1); here, $m = 5$. The weight w_{ji} is on the i-th

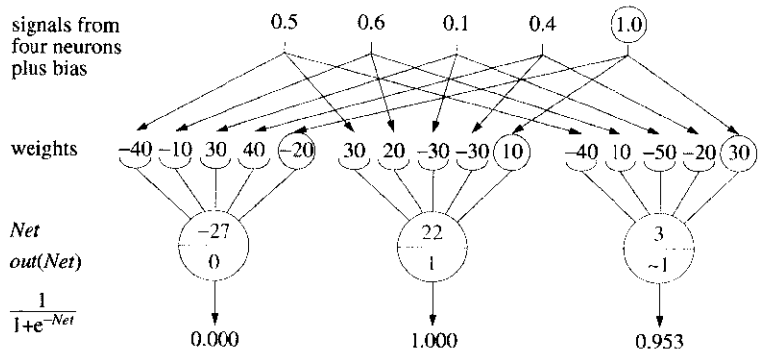

Figure 3-4: If the absolute values of the neuron weights are so large that the produced *Net* values are larger than +10, then the output is almost binary.

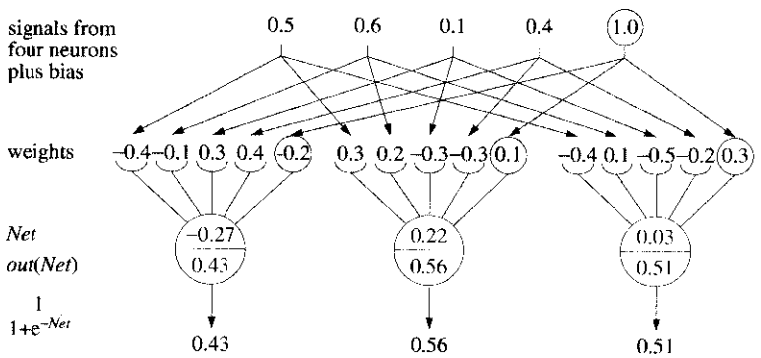

Figure 3-5: An example of a layer consisting of three neurons, each having five weights. The weight called *bias* is marked as a full circle; the value of the input signal to it is always 1.

position of the j-th neuron; for example, w_{23} is the 3rd weight of the 2nd neuron – see the circle above the second output neuron. The rightmost weight on each neuron (full circles) is the bias.

To make the picture unambiguous, a small example is given in Figure 3-4. A layer of three neurons is shown receiving signals from four neurons. For such an arrangement there must be five weights in each neuron. Remember, the input signal is actually 4-variate; the fifth input, which is always equal to 1, should be directed to the weight labeled *bias*. In order to check your understanding of the subject, you are encouraged to calculate the output values.

This example shows that the network responds continuously if the weights are small; it produces a continuum of values of the net input vector, ***Net***, between ±10 (see Table 3-1 and Table 2-2, first example, column 3). On the other hand, if the absolute values of the weights are much larger than 10, the network will act as a binary device, as we will now show.

Net	sf(*Net*)	
−10	0.0000	
−9	0.0001	
−8	0.0003	
−7	0.0011	
−6	0.0025	
−5	0.0067	
−4	0.0180	interval for giving values between 0 and 1, when the cut-off is set to 1% (0.01 or 0.99)
−3	0.0474	
−2	0.1192	
−1	0.3679	
0	0.500	
1	0.6321	
2	0.8808	
3	0.9526	
4	0.9820	
5	0.9937	
6	0.9975	
7	0.9989	
8	0.9997	
9	0.9999	
10	1.0000	

Table 3-1: Outputs of the sigmoidal transfer function (2.30) for different *Net* values

Let us submit the same set of four inputs, that is, the 5-variate input vector $\boldsymbol{X}$ = (0.5, 0.6, 0.1, 0.4, 1.0) shown in Figure 3-4, to the same neural network having weights larger than 10 or smaller than −10 (the weights of the previous example, −0.5 to +0.4, multiplied by 100).

Figure 3-5 shows that now the outputs of the neurons are either very close to 0 or very close to 1, quite in contrast to the results shown in Figure 3-4.

To get a feeling of how large the values of *Net* should be to obtain nonbinary output, some values of *sf*(*Net*) for $-10 \leq Net \leq 10$ are given in Table 3-1. These data illustrate why one usually selects very small random numbers for the initial weights, usually in the range of 0.1 or even smaller. A rule of thumb is to set the m initial values of the weights w_{ji} in each neuron j so that

$$\sum_{i=1}^{m} |w_{ji}| = 1 \tag{3.1}$$

The example corresponding to Figure 3-5 is important because it shows that in certain circumstances it is extremely difficult (and will take a large number of iterations) to change the output of the network with **small** corrections of weights, since the outputs will **discontinuously** flip between 0 and 1.

3.3 Input

Until now we did not bother much about the mechanism by which the signals actually enter the network. The fact that each signal x_i of the input vector $\boldsymbol{X}$ should come to all neurons in the first layer means that somehow x_i should be "distributed" over as many weights as there are neurons in the layer. Graphically this is shown in Figure 3-6.

In order to make the flow of input data graphically consistent with the flow of data within and between the layers of neurons, the crossing points in the top row of Figure 3-6a where each input signal x_i is forked towards the weights should be considered "neurons". These are the small circles in Figure 3-6b; on the output side, they behave as full-fledged neurons, able to send many signals of the same value to their attached neurons, but on the other side, each has only one input signal, x_i.

Nor do these "input neurons" change the input signals x_i at all, which means that they have neither weights nor any kind of transfer function. The "input neurons" only serve as distributors of signals and do not play any active role in modifying them.

In order to stress this difference between the non-active input "neurons" and the active ones, the former will be drawn throughout this book as in Figure 3-6c, as squares (and not as circles as many other authors do). In addition, we will refer to them as input *units* and not as input neurons.

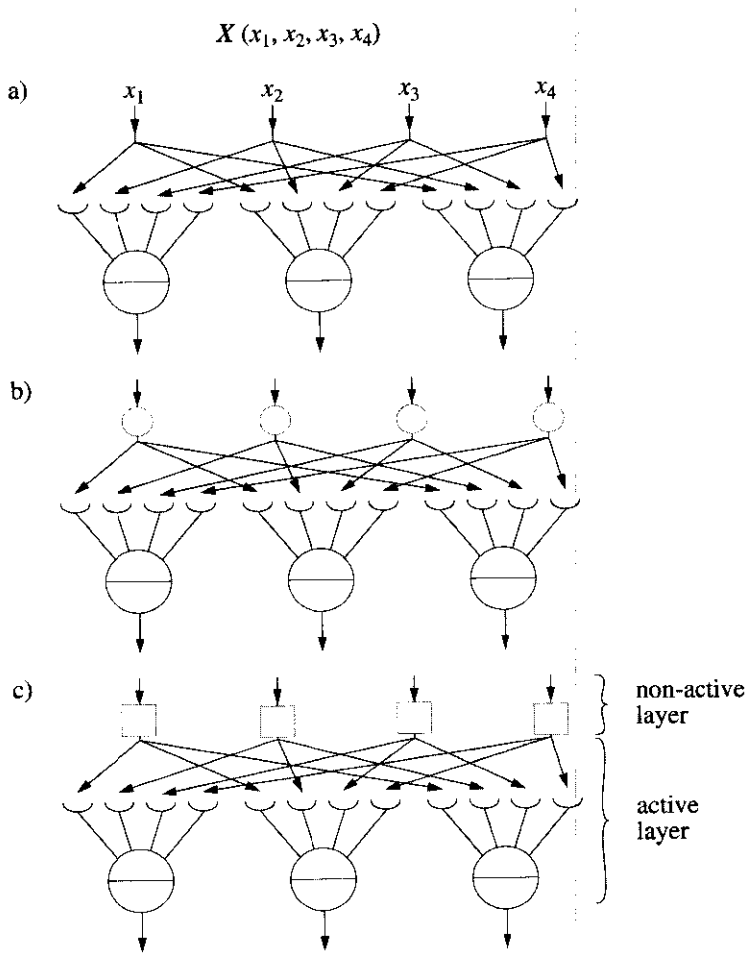

Figure 3-6:
a) Distribution of signals to the neurons in the first active layer.
(b) The points where the input signals are forked towards the neurons in the active layer are marked with small circles.
(c) Representing a non-active layer of neurons with squares.

> Be very careful about labeling the non-active input, active layer, and output layers of neurons. Be sure you know exactly which layer of neurons the author intends when he or she applies the terms "inputs" or "outputs" to a particular layer. Many times in the same equation the inputs and outputs are taken from different layers.

In **counting** the number of layers to classify the architecture of a network, we do not include the input layer.

3.4 Architectures

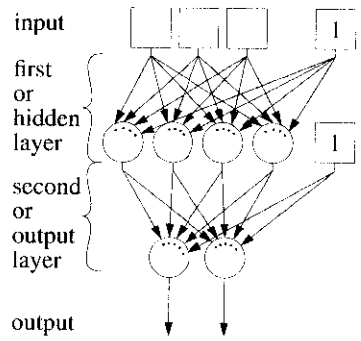

Figure 3-7: Linking layers of neurons together: two-layer design with input layer, one hidden and one output layer. Note that the number of weights in each neuron in a given layer corresponds to the number of signals produced by the layer above it.

The basic operation of a neuron is always the same: it collects a net input, Net_j, and transforms it into the output signal, out_j, via one of the transfer functions; the only thing we have to choose in advance is the number of layers, and the number of neurons in each layer.

All the topological data about the network:

- the number of inputs and outputs
- the number of layers
- the number of neurons in each layer
- the number of weights in each neuron
- the way the weights are linked together within or between the layer(s)
- which neurons receive the correction signals

together form the *architecture* or *design* of the network. Almost any network that is described as a "connected graph" can be said to have a neural network *architecture*; however, the acceptability of a given network architecture is judged solely by the result it produces.

In spite of such wide-open options for designing a neural network, there are some commonly accepted guidelines or restrictions. These are not imposed because of some fundamental requirements, but simply because theoretical investigations can be carried out much more easily if the network has an imposed order of design rather than random connections. The same is true for the problems that may appear at the programming stage.

The most common features of neural networks will be described briefly in the next paragraphs.

All neurons in one layer should obtain the same number of inputs, including an additional input connected to the bias. The number of weights in each neuron is fixed by the number of signals produced in the layer above it (Figure 3-7). (The network in this figure is referred to as a (3 x 4 x 2) network; the bias is not connected in this case.)

3.5 Hidden Layer; Output Layer

The layers below the passive input layer are usually referred to as the *hidden* layers, because they are not directly connected to the

outside world as the input units and the output neurons are. The layer of neurons that yields the final signal(s) is called the *output* layer. For now, we will simplify the figure of a neuron to a plain circle; you can think of weights as being distributed over the upper half arc of the circle.

In more complex neural network designs some of the signals might be input to neurons in more than one layer, some of which may lie much deeper in the network. The adjacent hidden layer might even be skipped altogether for some signals. Figure 3-8 shows both these possibilities in a multilayer neural network design: all neurons in the second layer get an additional signal from the input, and one neuron in the third layer gets a signal directly from the input (these links are drawn with thicker lines). Without them, this would be an ordinary three-layer (2 x 3 x 2 x 2) neural network.

In some cases, such "far-going" signals can be linked to only a few neurons on a particular deeper layer or only to a single one. This might cause problems in the correction algorithms because the weights are usually corrected layer by layer.

On linear computers, the nonstandard links cause a considerable slowdown of the computation because either the branching conditions have to be checked at each step, or some additional pointers to the proper weights have to be introduced.

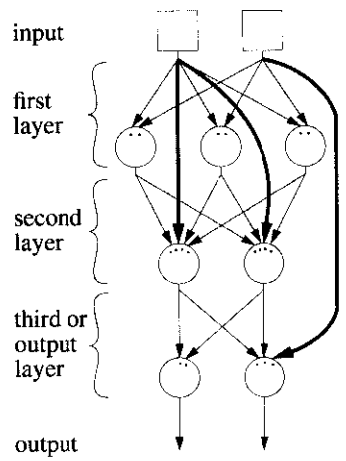

Figure 3-8: Three-layer neural network design with some signals bypassing the layer immediately below and linking to a deeper layer.

3.6 Graphical Representation of Neural Networks

Until now neural networks, consisting of layers of interconnected neurons have been shown as lines of circles (layers), linked together with arrows going from circles of one layer to the next. The arrows represent the direction of flow of the signals; at the places where the arrows touch the circles, the weights (synapses) are applied to them. The circles represent the bodies of the neurons, where all incoming signals are summed and the result then (nonlinearly) transformed into one output signal, which is transmitted to all neurons in the next layer.

Although it contains some simplifications of how neurons really work, this picture is quite adequate for describing artificial neural networks (e.g. Figure 3-7). It works, for example, whether the signal transformation actually takes place within the neuron's body or within the axon.

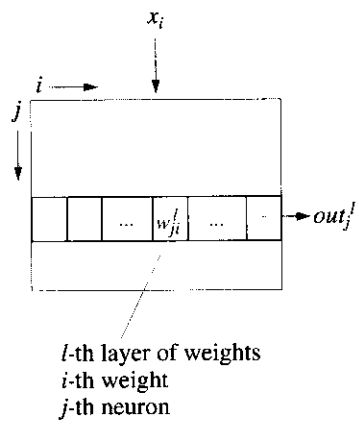

Figure 3-9: Matrix model of a network: the position of the weights within the neuron and the neurons within the layer.

However, it is based on a model of biological neurons; for programming, a matrix representation would be more explicit and precise.

The matrix representation considers a layer of n neurons, each having m weights, as an (n x m)-variate weight matrix $\boldsymbol{W}$. For a multilayer network, each matrix (and thus its elements w_{ji}) obtain a superscript l (l for layer) specifying the index of the layer. The notation:

$$w_{ji}^{l} \tag{3.2}$$

refers to the i-th weight of the j-th neuron in the l-th layer (Figure 3-9).

In the matrix notation, the input layer is actually a unit vector (a vector having all components equal to 1), and we do not need to draw or show it at all (if for some reason we do, then the layer index has to be 0). The weight matrix of the input layer, $\boldsymbol{W}^0$ that "transmits" m signals is a vector containing the value 1, n times:

$$\boldsymbol{W}^0 = (1, 1, 1, ..., 1) \tag{3.3}$$

in agreement with the fact that the weight matrix for the first active layer of weights $\boldsymbol{W}^1$ has a superscript of 1. It is evident that the last (the output) layer will therefore always have a layer index equal to the number of active layers in the network. This notation avoids a lot of confusion regarding the actual number of layers used in the given application.

The matrix notation shows clearly that the input signals to the l-th layer $\boldsymbol{X}^l$ and output signals $\boldsymbol{Out}^l$ from this layer are m- and n-dimensional vectors, respectively. As mentioned above we must remember that:

$$\begin{aligned} \boldsymbol{X}^l &= \boldsymbol{Out}^{l-1} \\ \text{or:} & \\ \boldsymbol{Out}^l &= \boldsymbol{X}^{l+1} \end{aligned} \tag{3.4}$$

To apply an m-variate signal input to a one-layer neural network consisting of n neurons each having m weights, we multiply an m-variate vector $\boldsymbol{X}(x_1, x_2, ..., x_{m-1}, 1)$ with the (n x m)-variate weight matrix $\boldsymbol{W}$. The result is an n-variate net input vector $\boldsymbol{Net}$ (Net_1, Net_2, ..., Net_n).

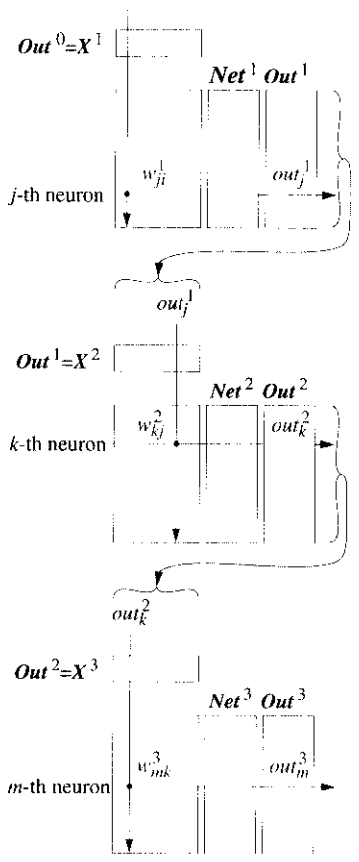

Figure 3-10: Matrix representation of a multilayer neural network. The number of weights in each layer is determined by the number of outputs from the layer above. However, the number of neurons in the layer is determined by the user (usually by trial and error).

$$\boldsymbol{Net} = (Net_1, Net_2, ..., Net_j, ..., Net_n) =$$

$$= \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ w_{31} & w_{32} & \cdots & w_{3m} \\ \vdots & \vdots & w_{ji} & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_{m-2} \\ x_{m-1} \\ 1 \end{bmatrix}$$

Using extended notation, we can show how each component Net_j is calculated for layer l:

$$Net_j^l = \sum_{i=1}^{m} w_{ji}^l x_i^l \qquad (3.5)$$

$$j = 1, 2, ..., n$$

The index j spans the n neurons, while i spans the m weights in the j-th neuron. The number of weights in the neuron is one more than the number of input variables, x_i; the remaining one input variable is the bias, which is always equal to 1.

The matrix equation (3.6) is a concise description of all net inputs in a one-layer network using the weight matrix $\boldsymbol{W}$:

$$\boldsymbol{Net}^l = \boldsymbol{W}^l \boldsymbol{X}^l \qquad (3.6)$$

In a multilayer network, the weight matrices representing the layers are distinguished by the superscript l. For such a calculation the l-th layer weight matrix elements w_{ji}^l are used together with the input to the l-th layer x_i^l. Because the input to the l-th layer is usually the output of the $(l-1)$-st layer, Equation (3.5) can be written as:

$$\boldsymbol{Net}^l = \boldsymbol{W}^l \boldsymbol{X}^l = \boldsymbol{W}^l \boldsymbol{Out}^{l-1}$$

or:

$$Net_j^l = \sum_{i=1}^{m} w_{ji}^l out_i^{l-1} \qquad (3.7)$$

$$j = 1, 2, ..., n$$

$\boldsymbol{Out}^l$ is obtained from $\boldsymbol{Net}^l$ by one of the transfer functions ((2.24), (2.25), (2.29), or (2.30)). Let us apply the sigmoidal function (2.30) here as an example:

$$\boldsymbol{Out}^l = \mathrm{sf}(\boldsymbol{Net}^l) \tag{3.8}$$

From Equations (3.4) to (3.7), it can be concluded that the input layer $\boldsymbol{W}^0$ (which has only one row of weights, all of them equal to 1 and has no transfer function) will produce an output $\boldsymbol{Out}^0$ that is exactly equal to the external input. Thus, the m-variate input vector $\boldsymbol{X}$ that is input to the network can be labeled as output also:

$$\boldsymbol{X}^1 = \boldsymbol{Out}^0 \tag{3.9}$$

Some authors draw networks so that information flows from left to right (Figure 3-11), and some, from bottom to top (Figure 3-12); but Kohonen favors (as do we, cf. Figure 3-7) the "top-down" design, which means that any input is above the neuron and its output is below. This corresponds to our everyday concept of "flow", whether in a liquid, a signal, or information. However, it must be said that most authors today prefer the "bottom-up" design (see Rumelhart's example, Figure 2-21d). This presumably originated with the convention in information theory where the flow of signals starts at the bottom. (If you lay the drawing in front of you on your desk, it runs away from you.)

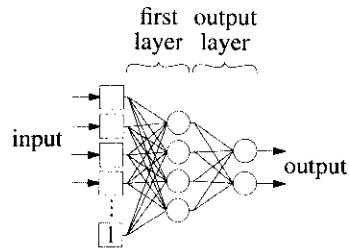

Figure 3-11: Neural network showing the flow of signals from left to right.

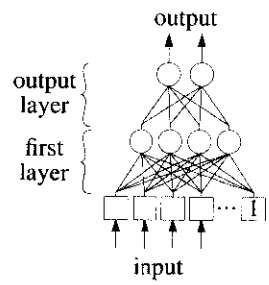

Figure 3-12: Neural network showing the flow of information from bottom to top.

3.7 Essentials

- **selection of weights**

$$\sum_{i=1}^{m} |w_{ji}| = 1 \tag{3.1}$$

- **indices of elements of the weight matrix**

w_{ji}^{l} — l-th layer, j-th neuron, i-th weight (3.2)

- **evaluation of the net input**

$$Net_j^l = \sum_{i=1}^{m} w_{ji}^l x_i^l \qquad j = 1, 2, ..., n \tag{3.5}$$

$$\boldsymbol{Out}^l = \mathrm{sf}(\boldsymbol{Net}^l) \tag{3.8}$$

$$Net_j^l = \sum_{i=1}^{m} w_{ji}^l out_i^{l-1} \qquad j = 1, 2, ..., n \tag{3.7}$$

- **evaluation of the output**

$$\boldsymbol{Out}^l = \mathrm{sf}(\boldsymbol{Net}^l) \tag{3.8}$$

- **labeling inputs and outputs**

$\boldsymbol{X}^l = \boldsymbol{Out}^{l-1}$ input to the l-th layer

or:

$\boldsymbol{Out}^l = \boldsymbol{X}^{l+1}$ output from the l-th layer (3.4)

($\boldsymbol{Out}^0$ is identical to the network input)

3.8 References and Suggested Readings

3-1. J. A. Anderson and E. Rosenfeld, Eds., *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, MA, USA, 1988, pp. 159.

3-2. M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, USA, 1969.

3-3. L. B. Elliot, "Neural Networks – Conference Update and Overview", *IEEE Expert*, Winter 1987, 12 – 13.

3-4. R. P. Lippmann, "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine*, April 1987, 4 – 22.

3-5. J. Zupan and J. Gasteiger, "Neural Networks: A New Method for Solving Chemical Problems or Just a Passing Phase?", *Anal. Chim. Acta* **248** (1991) 1 – 30.

3-6. J. Dayhoff, *Neural Network Architectures, An Introduction*, Van Nostrand Reinhold, New York, USA, 1990.

3-7. P. Smith-Churchland, *Neurophysiology: Towards a Unified Science of the Mind-Brain*, MIT Press, Cambridge, MA, USA, 1986, Chapter 2.