2 Neuron Model and Network Architectures

Objectives

In Chapter 1 we presented a simplified description of biological neurons and neural networks. Now we will introduce our simplified mathematical model of the neuron and will explain how these artificial neurons can be interconnected to form a variety of network architectures. We will also illustrate the basic operation of these networks through some simple examples. The concepts and notation introduced in this chapter will be used throughout this book.

This chapter does not cover all of the architectures that will be used in this book, but it does present the basic building blocks. More complex architectures will be introduced and discussed as they are needed in later chapters. Even so, a lot of detail is presented here. Please note that it is not necessary for the reader to memorize all of the material in this chapter on a first reading. Instead, treat it as a sample to get you started and a resource to which you can return.

Theory and Examples

Notation

Unfortunately, there is no single neural network notation that is universally accepted. Papers and books on neural networks have come from many diverse fields, including engineering, physics, psychology and mathematics, and many authors tend to use vocabulary peculiar to their specialty. As a result, many books and papers in this field are difficult to read, and concepts are made to seem more complex than they actually are. This is a shame, as it has prevented the spread of important new ideas. It has also led to more than one "reinvention of the wheel."

In this book we have tried to use standard notation where possible, to be clear and to keep matters simple without sacrificing rigor. In particular, we have tried to define practical conventions and use them consistently.

Figures, mathematical equations and text discussing both figures and mathematical equations will use the following notation:

Scalars — small *italic* letters: a,b,c

Vectors — small **bold** nonitalic letters: $\mathbf{a},\mathbf{b},\mathbf{c}$

Matrices — capital **BOLD** nonitalic letters: $\mathbf{A},\mathbf{B},\mathbf{C}$

Additional notation concerning the network architectures will be introduced as you read this chapter. A complete list of the notation that we use throughout the book is given in Appendix B, so you can look there if you have a question.

Neuron Model

Single-Input Neuron

Weight
Bias
Net Input
Transfer Function

A single-input neuron is shown in Figure 2.1. The scalar input p is multiplied by the scalar *weight* w to form wp, one of the terms that is sent to the summer. The other input, 1, is multiplied by a *bias* b and then passed to the summer. The summer output n, often referred to as the *net input*, goes into a *transfer function* f, which produces the scalar neuron output a. (Some authors use the term "activation function" rather than *transfer function* and "offset" rather than *bias*.)

If we relate this simple model back to the biological neuron that we discussed in Chapter 1, the weight w corresponds to the strength of a synapse, the cell body is represented by the summation and the transfer function, and the neuron output a represents the signal on the axon.

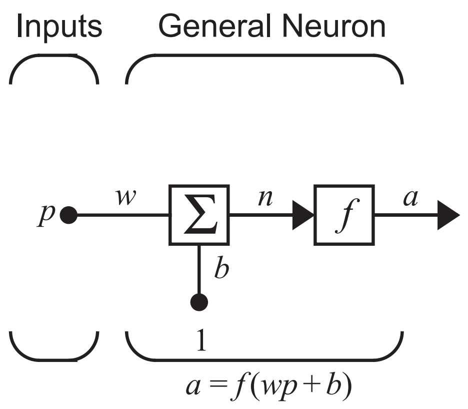


Figure 2.1 Single-Input Neuron

The neuron output is calculated as

$$a = f(wp + b) .$$

If, for instance, $w = 3$, $p = 2$ and $b = -1.5$, then

$$a = f(3(2) - 1.5) = f(4.5)$$

The actual output depends on the particular transfer function that is chosen. We will discuss transfer functions in the next section.

The bias is much like a weight, except that it has a constant input of 1. However, if you do not want to have a bias in a particular neuron, it can be omitted. We will see examples of this in Chapters 3, 7 and 16.

Note that w and b are both *adjustable* scalar parameters of the neuron. Typically the transfer function is chosen by the designer and then the parameters w and b will be adjusted by some learning rule so that the neuron input/output relationship meets some specific goal (see Chapter 4 for an introduction to learning rules). As described in the following section, we have different transfer functions for different purposes.

Transfer Functions

The transfer function in Figure 2.1 may be a linear or a nonlinear function of n. A particular transfer function is chosen to satisfy some specification of the problem that the neuron is attempting to solve.

A variety of transfer functions have been included in this book. Three of the most commonly used functions are discussed below.

Hard Limit Transfer Function

The *hard limit transfer function*, shown on the left side of Figure 2.2, sets the output of the neuron to 0 if the function argument is less than 0, or 1 if its argument is greater than or equal to 0. We will use this function to create neurons that classify inputs into two distinct categories. It will be used extensively in Chapter 4.

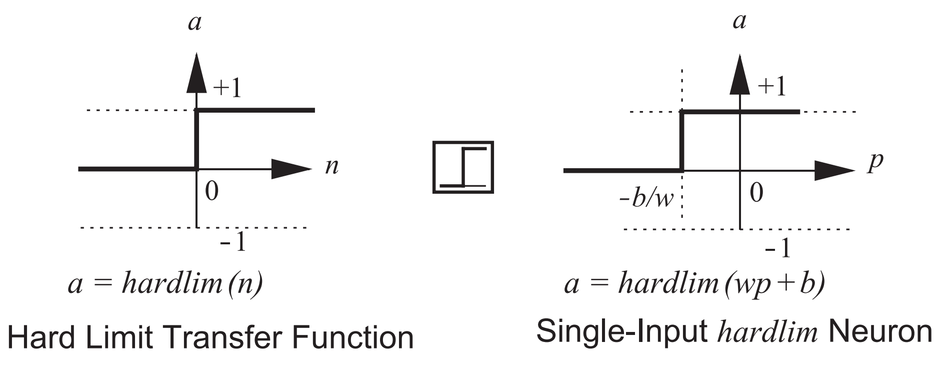


Figure 2.2 Hard Limit Transfer Function

The graph on the right side of Figure 2.2 illustrates the input/output characteristic of a single-input neuron that uses a hard limit transfer function. Here we can see the effect of the weight and the bias. Note that an icon for the hard limit transfer function is shown between the two figures. Such icons will replace the general f in network diagrams to show the particular transfer function that is being used.

Linear Transfer Function

The output of a *linear transfer function* is equal to its input:

$$a = n , \tag{2.1}$$

as illustrated in Figure 2.3.

Neurons with this transfer function are used in the ADALINE networks, which are discussed in Chapter 10.

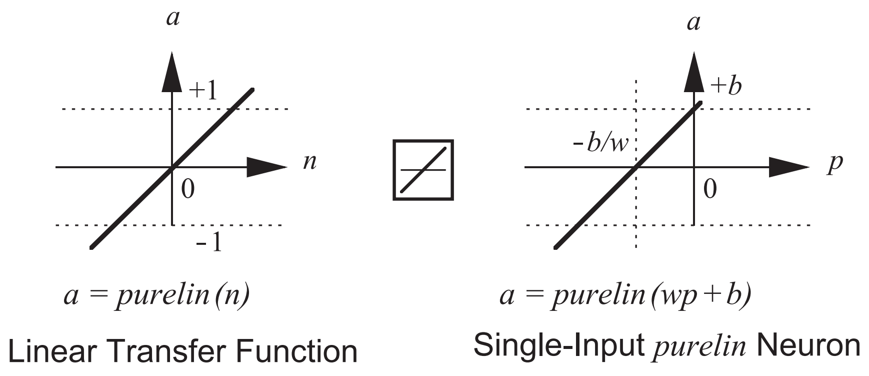


Figure 2.3 Linear Transfer Function

The output (a) versus input (p) characteristic of a single-input linear neuron with a bias is shown on the right of Figure 2.3.

Log-Sigmoid Transfer Function

The *log-sigmoid transfer function* is shown in Figure 2.4.

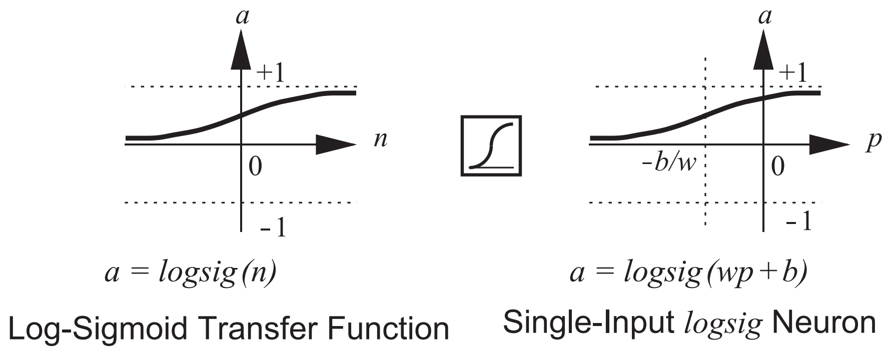


Figure 2.4 Log-Sigmoid Transfer Function

This transfer function takes the input (which may have any value between plus and minus infinity) and squashes the output into the range 0 to 1, according to the expression:

$$a = \frac{1}{1 + e^{-n}}. \tag{2.2}$$

The log-sigmoid transfer function is commonly used in multilayer networks that are trained using the backpropagation algorithm, in part because this function is differentiable (see Chapter 11).

Most of the transfer functions used in this book are summarized in Table 2.1. Of course, you can define other transfer functions in addition to those shown in Table 2.1 if you wish.

To experiment with a single-input neuron, use the Neural Network Design Demonstration One-Input Neuron `nnd2n1`.

Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1 + e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons	C	compet

Table 2.1 Transfer Functions

Multiple-Input Neuron

Weight Matrix

Typically, a neuron has more than one input. A neuron with R inputs is shown in Figure 2.5. The individual inputs $p_1, p_2, \ldots, p_R$ are each weighted by corresponding elements $w_{1,1}, w_{1,2}, \ldots, w_{1,R}$ of the *weight matrix* $\mathbf{W}$.

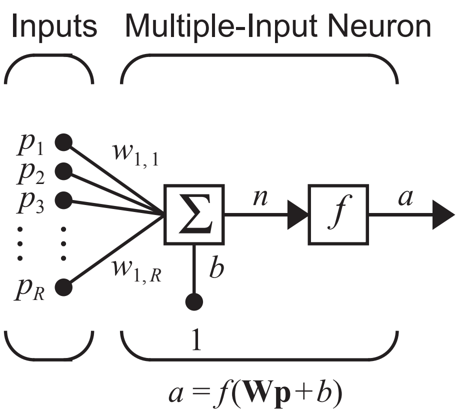


Figure 2.5 Multiple-Input Neuron

The neuron has a bias b, which is summed with the weighted inputs to form the net input n:

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \cdots + w_{1,R}p_R + b. \tag{2.3}$$

This expression can be written in matrix form:

$$n = \mathbf{Wp} + b, \tag{2.4}$$

where the matrix $\mathbf{W}$ for the single neuron case has only one row.

Now the neuron output can be written as

$$a = f(\mathbf{Wp} + b). \tag{2.5}$$

Fortunately, neural networks can often be described with matrices. This kind of matrix expression will be used throughout the book. Don't be concerned if you are rusty with matrix and vector operations. We will review these topics in Chapters 5 and 6, and we will provide many examples and solved problems that will spell out the procedures.

Weight Indices

We have adopted a particular convention in assigning the indices of the elements of the weight matrix. The first index indicates the particular neuron destination for that weight. The second index indicates the source of the signal fed to the neuron. Thus, the indices in $w_{1,2}$ say that this weight represents the connection *to* the first (and only) neuron *from* the second source. Of course, this convention is more useful if there is more than one neuron, as will be the case later in this chapter.

We would like to draw networks with several neurons, each having several inputs. Further, we would like to have more than one layer of neurons. You can imagine how complex such a network might appear if all the lines were drawn. It would take a lot of ink, could hardly be read, and the mass of detail might obscure the main features. Thus, we will use an *abbreviated notation*. A multiple-input neuron using this notation is shown in Figure 2.6.

Abbreviated Notation

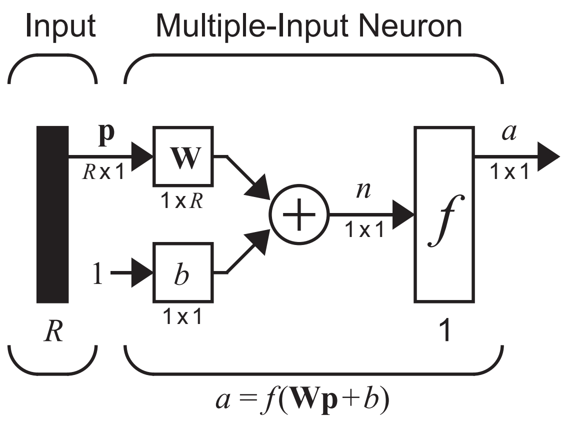


Figure 2.6 Neuron with R Inputs, Abbreviated Notation

As shown in Figure 2.6, the input vector $\mathbf{p}$ is represented by the solid vertical bar at the left. The dimensions of $\mathbf{p}$ are displayed below the variable as $R \times 1$, indicating that the input is a single vector of R elements. These inputs go to the weight matrix $\mathbf{W}$, which has R columns but only one row in this single neuron case. A constant 1 enters the neuron as an input and is multiplied by a scalar bias b. The net input to the transfer function f is n, which is the sum of the bias b and the product $\mathbf{W}\mathbf{p}$. The neuron's output a is a scalar in this case. If we had more than one neuron, the network output would be a vector.

The dimensions of the variables in these abbreviated notation figures will always be included, so that you can tell immediately if we are talking about a scalar, a vector or a matrix. You will not have to guess the kind of variable or its dimensions.

Note that the number of inputs to a network is set by the external specifications of the problem. If, for instance, you want to design a neural network that is to predict kite-flying conditions and the inputs are air temperature, wind velocity and humidity, then there would be three inputs to the network.

To experiment with a two-input neuron, use the Neural Network Design Demonstration Two-Input Neuron (`nnd2n2`).

Network Architectures

Commonly one neuron, even with many inputs, may not be sufficient. We might need five or ten, operating in parallel, in what we will call a "layer." This concept of a layer is discussed below.

A Layer of Neurons

Layer

A single-*layer* network of S neurons is shown in Figure 2.7. Note that each of the R inputs is connected to each of the neurons and that the weight matrix now has S rows.

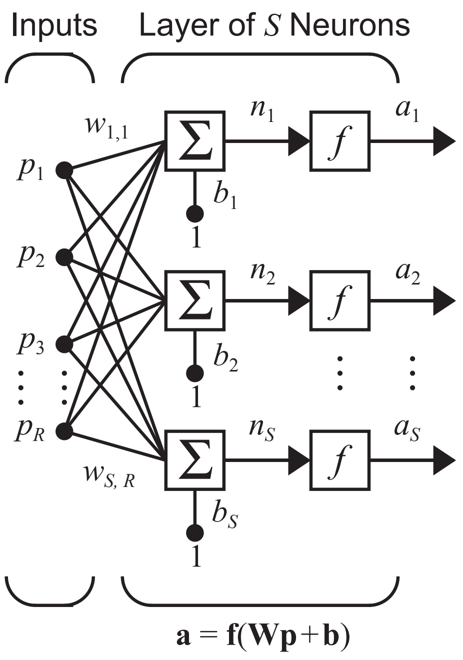


Figure 2.7 Layer of S Neurons

The layer includes the weight matrix, the summers, the bias vector $\mathbf{b}$, the transfer function boxes and the output vector $\mathbf{a}$. Some authors refer to the inputs as another layer, but we will not do that here.

Each element of the input vector $\mathbf{p}$ is connected to each neuron through the weight matrix $\mathbf{W}$. Each neuron has a bias b_i, a summer, a transfer function f and an output a_i. Taken together, the outputs form the output vector $\mathbf{a}$.

It is common for the number of inputs to a layer to be different from the number of neurons (i.e., $R \neq S$).

You might ask if all the neurons in a layer must have the same transfer function. The answer is no; you can define a single (composite) layer of neurons having different transfer functions by combining two of the networks shown above in parallel. Both networks would have the same inputs, and each network would create some of the outputs.

The input vector elements enter the network through the weight matrix $\mathbf{W}$:

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \\ w_{2,1} & w_{2,2} & \dots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \dots & w_{S,R} \end{bmatrix}. \quad (2.6)$$

As noted previously, the row indices of the elements of matrix $\mathbf{W}$ indicate the destination neuron associated with that weight, while the column indices indicate the source of the input for that weight. Thus, the indices in $w_{3,2}$ say that this weight represents the connection *to* the third neuron *from* the second source.

Fortunately, the *S*-neuron, *R*-input, one-layer network also can be drawn in abbreviated notation, as shown in Figure 2.8.

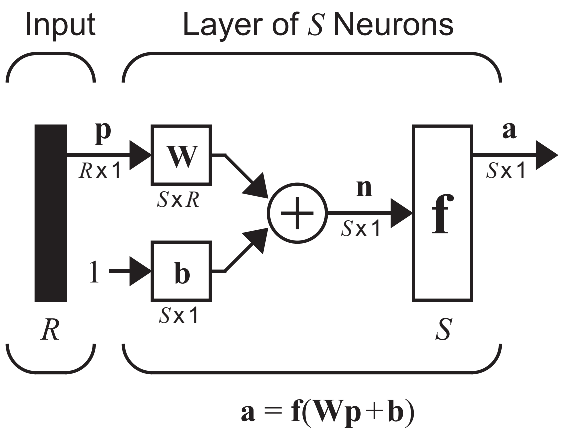


Figure 2.8 Layer of S Neurons, Abbreviated Notation

Here again, the symbols below the variables tell you that for this layer, $\mathbf{p}$ is a vector of length R, $\mathbf{W}$ is an $S \times R$ matrix, and $\mathbf{a}$ and $\mathbf{b}$ are vectors of length S. As defined previously, the layer includes the weight matrix, the summation and multiplication operations, the bias vector $\mathbf{b}$, the transfer function boxes and the output vector.

Multiple Layers of Neurons

Now consider a network with several layers. Each layer has its own weight matrix $\mathbf{W}$, its own bias vector $\mathbf{b}$, a net input vector $\mathbf{n}$ and an output vector $\mathbf{a}$. We need to introduce some additional notation to distinguish between

Layer Superscript

these layers. We will use superscripts to identify the layers. Specifically, we append the number of the layer as a *superscript* to the names for each of these variables. Thus, the weight matrix for the first layer is written as $\mathbf{W}^1$, and the weight matrix for the second layer is written as $\mathbf{W}^2$. This notation is used in the three-layer network shown in Figure 2.9.

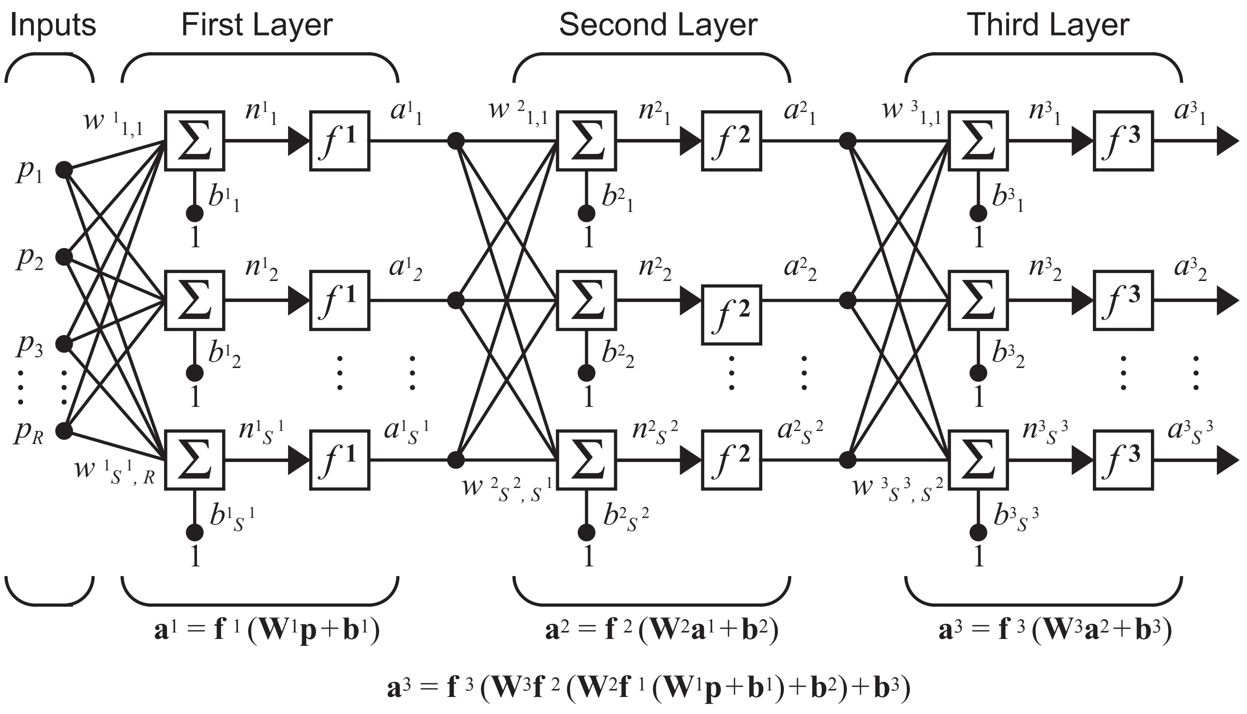


Figure 2.9 Three-Layer Network

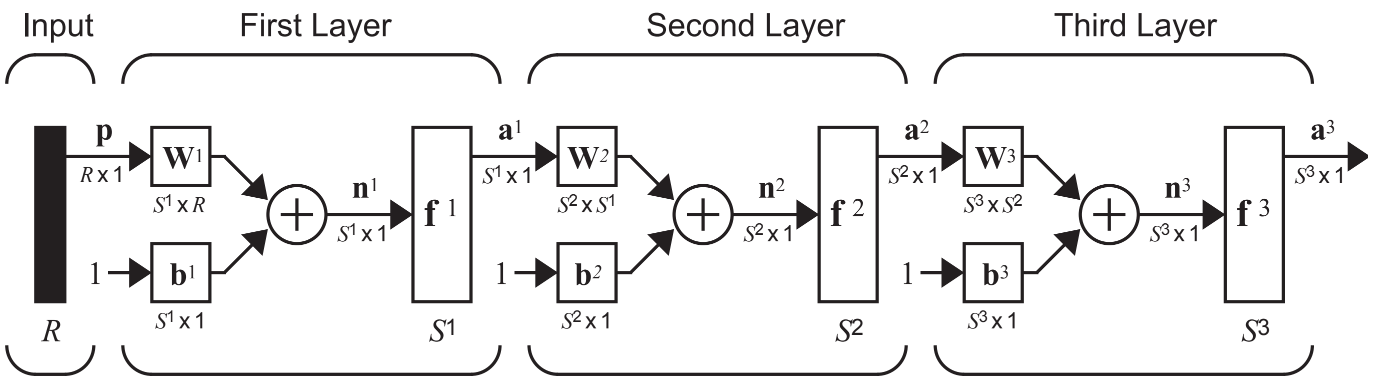
As shown, there are R inputs, S^1 neurons in the first layer, S^2 neurons in the second layer, etc. As noted, different layers can have different numbers of neurons.

The outputs of layers one and two are the inputs for layers two and three. Thus layer 2 can be viewed as a one-layer network with $R = S^1$ inputs, $S = S^2$ neurons, and an $S^2 \times S^1$ weight matrix $\mathbf{W}^2$. The input to layer 2 is $\mathbf{a}^1$, and the output is $\mathbf{a}^2$.

Output Layer
Hidden Layers

A layer whose output is the network output is called an *output layer*. The other layers are called *hidden layers*. The network shown above has an output layer (layer 3) and two hidden layers (layers 1 and 2).

The same three-layer network discussed previously also can be drawn using our abbreviated notation, as shown in Figure 2.10.



$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1) \qquad \mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1+\mathbf{b}^2) \qquad \mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2+\mathbf{b}^3)$$

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

Figure 2.10 Three-Layer Network, Abbreviated Notation

Multilayer networks are more powerful than single-layer networks. For instance, a two-layer network having a sigmoid first layer and a linear second layer can be trained to approximate most functions arbitrarily well. Single-layer networks cannot do this.

At this point the number of choices to be made in specifying a network may look overwhelming, so let us consider this topic. The problem is not as bad as it looks. First, recall that the number of inputs to the network and the number of outputs from the network are defined by external problem specifications. So if there are four external variables to be used as inputs, there are four inputs to the network. Similarly, if there are to be seven outputs from the network, there must be seven neurons in the output layer. Finally, the desired characteristics of the output signal also help to select the transfer function for the output layer. If an output is to be either -1 or 1, then a symmetrical hard limit transfer function should be used. Thus, the architecture of a single-layer network is almost completely determined by problem specifications, including the specific number of inputs and outputs and the particular output signal characteristic.

Now, what if we have more than two layers? Here the external problem does not tell you directly the number of neurons required in the hidden layers. In fact, there are few problems for which one can predict the optimal number of neurons needed in a hidden layer. This problem is an active area of research. We will develop some feeling on this matter as we proceed to Chapter 11, Backpropagation.

As for the number of layers, most practical neural networks have just two or three layers. Four or more layers are used rarely.

We should say something about the use of biases. One can choose neurons with or without biases. The bias gives the network an extra variable, and so you might expect that networks with biases would be more powerful than those without, and that is true. Note, for instance, that a neuron without a bias will always have a net input n of zero when the network inputs $\mathbf{p}$ are zero. This may not be desirable and can be avoided by the use of a bias. The effect of the bias is discussed more fully in Chapters 3, 4 and 5.

In later chapters we will omit a bias in some examples or demonstrations. In some cases this is done simply to reduce the number of network parameters. With just two variables, we can plot system convergence in a two-dimensional plane. Three or more variables are difficult to display.

Recurrent Networks

Delay

Before we discuss recurrent networks, we need to introduce some simple building blocks. The first is the *delay* block, which is illustrated in Figure 2.11.

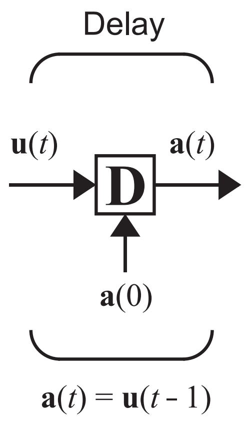


Figure 2.11 Delay Block

The delay output $\mathbf{a}(t)$ is computed from its input $\mathbf{u}(t)$ according to

$$\mathbf{a}(t) = \mathbf{u}(t-1) \,. \tag{2.7}$$

Thus the output is the input delayed by one time step. (This assumes that time is updated in discrete steps and takes on only integer values.) Eq. (2.7) requires that the output be initialized at time $t = 0$. This initial condition is indicated in Figure 2.11 by the arrow coming into the bottom of the delay block.

Integrator

Another related building block, which we will use for the continuous-time recurrent networks in Chapters 18–21, is the *integrator*, which is shown in Figure 2.12.

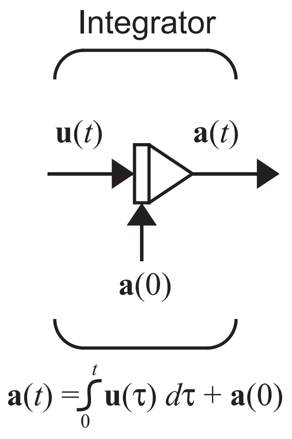

Integrator
$\mathbf{u}(t)$
$\mathbf{a}(t)$
$\mathbf{a}(0)$
$\mathbf{a}(t) = \int_0^t \mathbf{u}(\tau)\, d\tau + \mathbf{a}(0)$

Figure 2.12 Integrator Block

The integrator output $\mathbf{a}(t)$ is computed from its input $\mathbf{u}(t)$ according to

$$\mathbf{a}(t) = \int_0^t \mathbf{u}(\tau) d\tau + \mathbf{a}(0) . \tag{2.8}$$

The initial condition $\mathbf{a}(0)$ is indicated by the arrow coming into the bottom of the integrator block.

Recurrent Network

We are now ready to introduce recurrent networks. A *recurrent network* is a network with feedback; some of its outputs are connected to its inputs. This is quite different from the networks that we have studied thus far, which were strictly feedforward with no backward connections. One type of discrete-time recurrent network is shown in Figure 2.13.

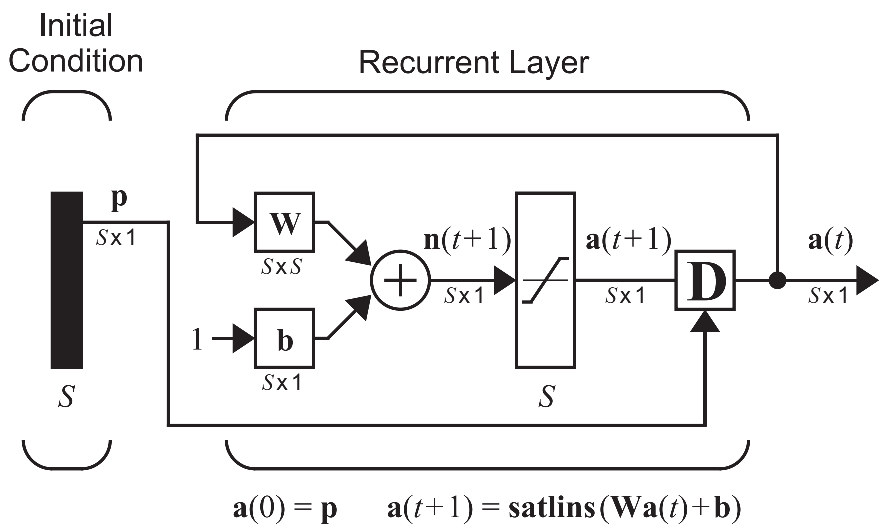


Figure 2.13 Recurrent Network

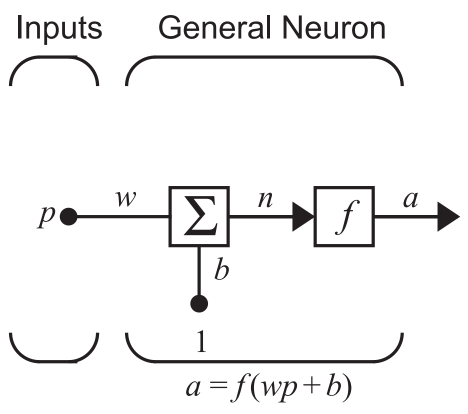
In this particular network the vector $\mathbf{p}$ supplies the initial conditions (i.e., $\mathbf{a}(0) = \mathbf{p}$). Then future outputs of the network are computed from previous outputs:

$$\mathbf{a}(1) = \mathbf{satlins}(\mathbf{W}\mathbf{a}(0) + \mathbf{b}),\ \mathbf{a}(2) = \mathbf{satlins}(\mathbf{W}\mathbf{a}(1) + \mathbf{b}),\ \ldots$$

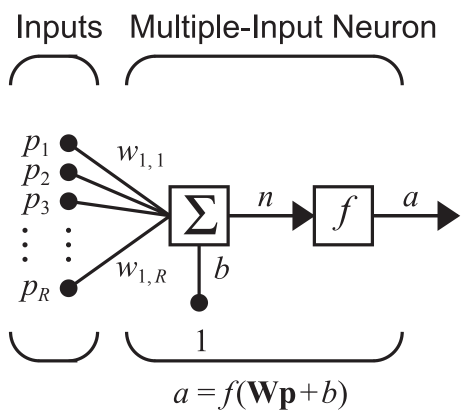
Recurrent networks are potentially more powerful than feedforward networks and can exhibit temporal behavior. These types of networks are discussed in Chapters 3, 14 and 18–21.

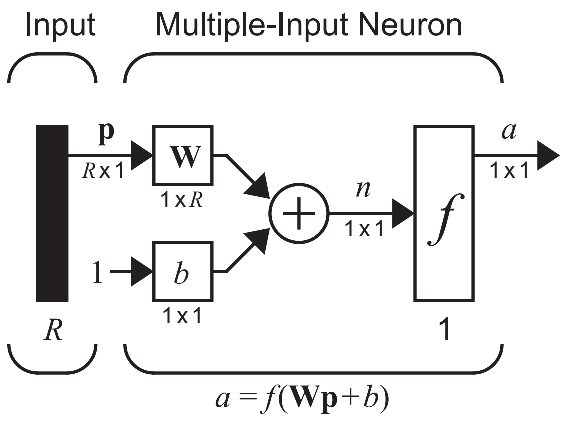
Summary of Results

Single-Input Neuron



Multiple-Input Neuron

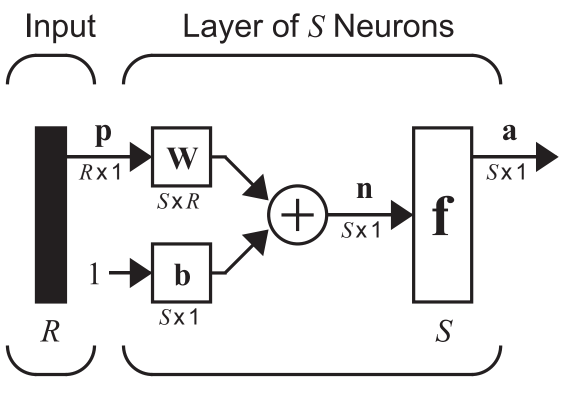




Transfer Functions

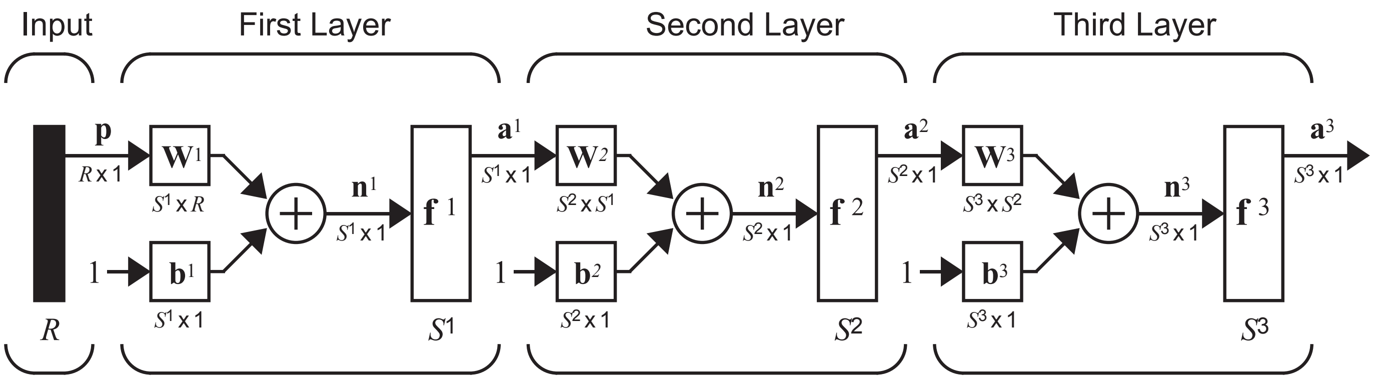
Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$	_|‾	hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$	_|‾	hardlims
Linear	$a = n$	/	purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$	_/‾	satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$	_/‾	satlins
Log-Sigmoid	$a = \frac{1}{1 + e^{-n}}$	∫	logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$	∫	tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$	_/	poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons	$\boxed{C}$	compet

Layer of Neurons



$\mathbf{a} = \mathbf{f}(\mathbf{Wp}+\mathbf{b})$

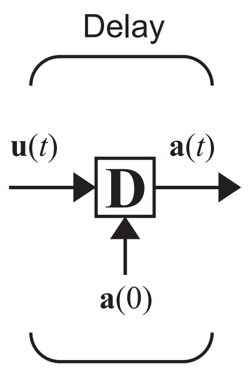
Three Layers of Neurons



$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)$ $\mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1+\mathbf{b}^2)$ $\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2+\mathbf{b}^3)$

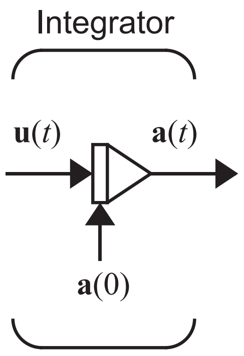
$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$

Delay



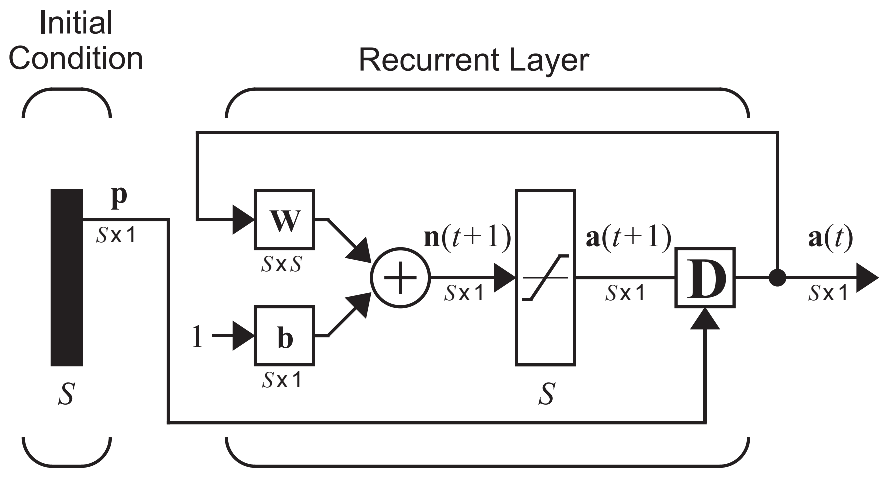
$\mathbf{a}(t) = \mathbf{u}(t-1)$

Integrator



$$\mathbf{a}(t) = \int_0^t \mathbf{u}(\tau)\, d\tau + \mathbf{a}(0)$$

Recurrent Network



$$\mathbf{a}(0) = \mathbf{p} \qquad \mathbf{a}(t+1) = \mathbf{satlins}(\mathbf{Wa}(t)+\mathbf{b})$$

How to Pick an Architecture

Problem specifications help define the network in the following ways:

1. Number of network inputs = number of problem inputs
2. Number of neurons in output layer = number of problem outputs
3. Output layer transfer function choice at least partly determined by problem specification of the outputs

Solved Problems

P2.1 The input to a single-input neuron is 2.0, its weight is 2.3 and its bias is -3.

i. What is the net input to the transfer function?

ii. What is the neuron output?

i. The net input is given by:

$$n = wp + b = (2.3)(2) + (-3) = 1.6$$

ii. The output cannot be determined because the transfer function is not specified.

P2.2 What is the output of the neuron of P2.1 if it has the following transfer functions?

i. Hard limit

ii. Linear

iii. Log-sigmoid

i. For the hard limit transfer function:

$$a = hardlim(1.6) = 1.0$$

ii. For the linear transfer function:

$$a = purelin(1.6) = 1.6$$

iii. For the log-sigmoid transfer function:

$$a = logsig(1.6) = \frac{1}{1 + e^{-1.6}} = 0.8320$$

```
» 2 + 2
ans =
     4
```

Verify this result using MATLAB and the function `logsig`, which is in the MININNET directory (see Appendix B).

P2.3 Given a two-input neuron with the following parameters: $b = 1.2$**,** $\mathbf{W} = \begin{bmatrix} 3 & 2 \end{bmatrix}$ **and** $\mathbf{p} = \begin{bmatrix} -5 & 6 \end{bmatrix}^T$**, calculate the neuron output for the following transfer functions:**

i. A symmetrical hard limit transfer function

ii. A saturating linear transfer function

iii. A hyperbolic tangent sigmoid (tansig) transfer function

First calculate the net input n:

$$n = \mathbf{W}\mathbf{p} + b = \begin{bmatrix} 3 & 2 \end{bmatrix} \begin{bmatrix} -5 \\ 6 \end{bmatrix} + (1.2) = -1.8 .$$

Now find the outputs for each of the transfer functions.

i. $a = hardlims(-1.8) = -1$

ii. $a = satlin(-1.8) = 0$

iii. $a = tansig(-1.8) = -0.9468$

P2.4 A single-layer neural network is to have six inputs and two outputs. The outputs are to be limited to and continuous over the range 0 to 1. What can you tell about the network architecture? Specifically:

i. How many neurons are required?

ii. What are the dimensions of the weight matrix?

iii. What kind of transfer functions could be used?

iv. Is a bias required?

The problem specifications allow you to say the following about the network.

i. Two neurons, one for each output, are required.

ii. The weight matrix has two rows corresponding to the two neurons and six columns corresponding to the six inputs. (The product $\mathbf{W}\mathbf{p}$ is a two-element vector.)

iii. Of the transfer functions we have discussed, the $logsig$ transfer function would be most appropriate.

iv. Not enough information is given to determine if a bias is required.

Epilogue

This chapter has introduced a simple artificial neuron and has illustrated how different neural networks can be created by connecting groups of neurons in various ways. One of the main objectives of this chapter has been to introduce our basic notation. As the networks are discussed in more detail in later chapters, you may wish to return to Chapter 2 to refresh your memory of the appropriate notation.

This chapter was not meant to be a complete presentation of the networks we have discussed here. That will be done in the chapters that follow. We will begin in Chapter 3, which will present a simple example that uses some of the networks described in this chapter, and will give you an opportunity to see these networks in action. The networks demonstrated in Chapter 3 are representative of the types of networks that are covered in the remainder of this text.

Exercises

E2.1 A single input neuron has a weight of 1.3 and a bias of 3.0. What possible kinds of transfer functions, from Table 2.1, could this neuron have, if its output is given below. In each case, give the value of the input that would produce these outputs.

i. 1.6

ii. 1.0

iii. 0.9963

iv. -1.0

E2.2 Consider a single-input neuron with a bias. We would like the output to be -1 for inputs less than 3 and +1 for inputs greater than or equal to 3.

i. What kind of a transfer function is required?

ii. What bias would you suggest? Is your bias in any way related to the input weight? If yes, how?

iii. Summarize your network by naming the transfer function and stating the bias and the weight. Draw a diagram of the network. Verify the network performance using MATLAB.

E2.3 Given a two-input neuron with the following weight matrix and input vector: $\mathbf{W} = \begin{bmatrix} 3 & 2 \end{bmatrix}$ and $\mathbf{p} = \begin{bmatrix} -5 & 7 \end{bmatrix}^T$, we would like to have an output of 0.5. Do you suppose that there is a combination of bias and transfer function that might allow this?

i. Is there a transfer function from Table 2.1 that will do the job if the bias is zero?

ii. Is there a bias that will do the job if the linear transfer function is used? If yes, what is it?

iii. Is there a bias that will do the job if a log-sigmoid transfer function is used? Again, if yes, what is it?

iv. Is there a bias that will do the job if a symmetrical hard limit transfer function is used? Again, if yes, what is it?

E2.4 A two-layer neural network is to have four inputs and six outputs. The range of the outputs is to be continuous between 0 and 1. What can you tell about the network architecture? Specifically:

i. How many neurons are required in each layer?

ii. What are the dimensions of the first-layer and second-layer weight matrices?

iii. What kinds of transfer functions can be used in each layer?

iv. Are biases required in either layer?

E2.5 Consider the following neuron.

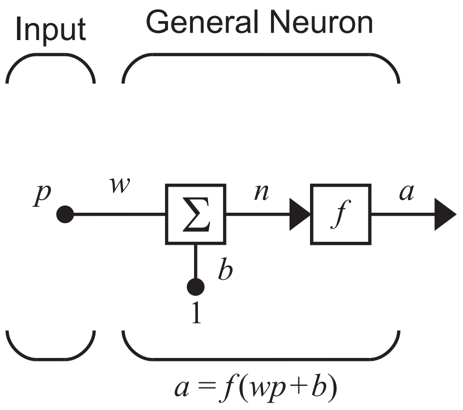


Figure P15.1 General Neuron

Sketch the neuron response (plot a versus p for $-2<p<2$) for the following cases.

i. $w = 1$, $b = 1$, $f = hardlims$.

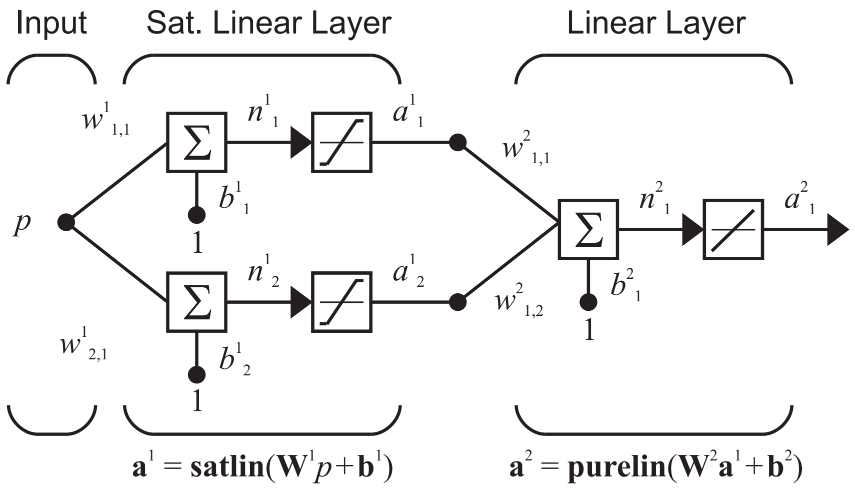
ii. $w = -1$, $b = 1$, $f = hardlims$.

iii. $w = 2$, $b = 3$, $f = purelin$.

iv. $w = 2$, $b = 3$, $f = satlins$.

v. $w = -2$, $b = -1$, $f = poslin$.

E2.6 Consider the following neural network.



$$w^1_{1,1} = 2,\ w^1_{2,1} = 1,\ b^1_1 = 2,\ b^1_2 = -1,\ w^2_{1,1} = 1,\ w^2_{1,2} = -1,\ b^2_1 = 0$$

Sketch the following responses (plot the indicated variable versus p for $-3 < p < 3$).

i. n^1_1.

ii. a^1_1.

iii. n^1_2

iv. a^1_2.

v. n^2_1.

vi. a^2_1.