15 Associative Learning

Objectives

The neural networks we have discussed so far (in Chapters 4, 7, 10–14) have all been trained in a supervised manner. Each network required a target signal to define correct network behavior.

In contrast, this chapter introduces a collection of simple rules that allow unsupervised learning. These rules give networks the ability to learn associations between patterns that occur together frequently. Once learned, associations allow networks to perform useful tasks such as pattern recognition and recall.

Despite the simplicity of the rules in this chapter, they will form the foundation for powerful networks in Chapters 16, 18, 19.

Theory and Examples

This chapter is all about associations: how associations can be represented by a network, how a network can learn new associations.

Stimulus
Response

What is an association? An association is any link between a system's input and output such that when a pattern A is presented to the system it will respond with pattern B. When two patterns are linked by an association, the input pattern is often referred to as the *stimulus*. Likewise, the output pattern is referred to as the *response*.

Associations are so fundamental that they formed the foundation of the behaviorist school of psychology. This branch of psychology attempted to explain much of animal and human behavior by using associations and rules for learning associations. (This approach has since been largely discredited.)

One of the earliest influences on the behaviorist school of psychology was the classic experiment of Ivan Pavlov, in which he trained a dog to salivate at the sound of a bell, by ringing the bell whenever food was presented. This is an example of what is now called classical conditioning. B. F. Skinner was one of the most influential proponents of the behaviorist school. His classic experiment involved training a rat to press a bar in order to obtain a food pellet. This is an example of instrumental conditioning.

It was to provide a biological explanation for some of this behavior that led Donald Hebb to his postulate, previously quoted in Chapter 7 [Hebb49]:

"*When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.*"

In Chapter 7 we analyzed the performance of a supervised learning rule based on Hebb's postulate. In this chapter we will discuss unsupervised forms of Hebbian learning, as well as other related associative learning rules.

A number of researchers have contributed to the development of associative learning. In particular, Tuevo Kohonen, James Anderson and Stephen Grossberg have been very influential. Anderson and Kohonen independently developed the linear associator network in the late 1960s and early 1970s ([Ande72], [Koho72]). Grossberg introduced nonlinear continuous-time associative networks during the same time period (e.g., [Gross68]). All of these researchers, in addition to many others, have continued the development of associative learning up to the present time.

In this chapter we will discuss the elemental associative learning rules. Then, in Chapters 14–16 we will present more complex networks that use

associative learning as a primary component. Chapter 16 will describe Kohonen networks, and Chapters 18 and 19 will discuss Grossberg networks.

Simple Associative Network

Let's take a look at the simplest network capable of implementing an association. An example is the single-input hard limit neuron shown in Figure 15.1.

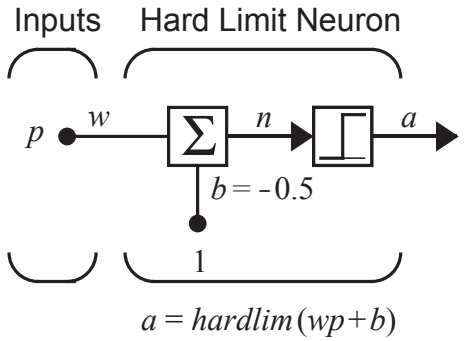


Figure 15.1 Single-Input Hard Limit Associator

The neuron's output a is determined from its input p according to

$$a = hardlim(wp + b) = hardlim(wp - 0.5). \tag{15.1}$$

For simplicity, we will restrict the value of p to be either 0 or 1, indicating whether a stimulus is absent or present. Note that a is limited to the same values by the hard limit transfer function. It indicates the presence or absence of the network's response.

$$p = \begin{cases} 1, & \text{stimulus} \\ 0, & \text{no stimulus} \end{cases} \qquad a = \begin{cases} 1, & \text{response} \\ 0, & \text{no response} \end{cases} \tag{15.2}$$

The presence of an association between the stimulus $p = 1$, and the response $a = 1$ is dictated by the value of w. The network will respond to the stimulus only if w is greater than $-b$ (in this case 0.5).

The learning rules discussed in this chapter are normally used in the framework of a larger network, such as the competitive networks of Chapters 16, 18 and 19. In order to demonstrate the operation of the associative learning rules, without using complex networks, we will use simple networks that have two types of inputs.

Unconditioned Stimulus

Conditioned Stimulus

One set of inputs will represent the *unconditioned stimulus*. This is analogous to the food presented to the dog in Pavlov's experiment. Another set of inputs will represent the *conditioned stimulus*. This is analogous to the bell in Pavlov's experiment. Initially the dog salivates only when food is

presented. This is an innate characteristic that does not have to be learned. However, when the bell is repeatedly paired with the food, the dog is conditioned to salivate at the sound of the bell, even when no food is present.

We will represent the unconditioned stimulus as $\mathbf{p}^0$ and the conditioned stimulus simply as $\mathbf{p}$. For our purposes we will assume that the weights associated with $\mathbf{p}^0$ are fixed, but that the weights associated with $\mathbf{p}$ are adjusted according to the relevant learning rule.

Figure 15.2 shows a network for recognizing bananas. The network has both an unconditioned stimulus (banana shape) and a conditioned stimulus (banana smell). We don't mean to imply here that smell is more conditionable than sight. In our examples in this chapter the choices of conditioned and unconditioned stimuli are arbitrary and are used simply to demonstrate the performance of the learning rules. We will use this network to demonstrate the operation of the Hebb rule in the following section.

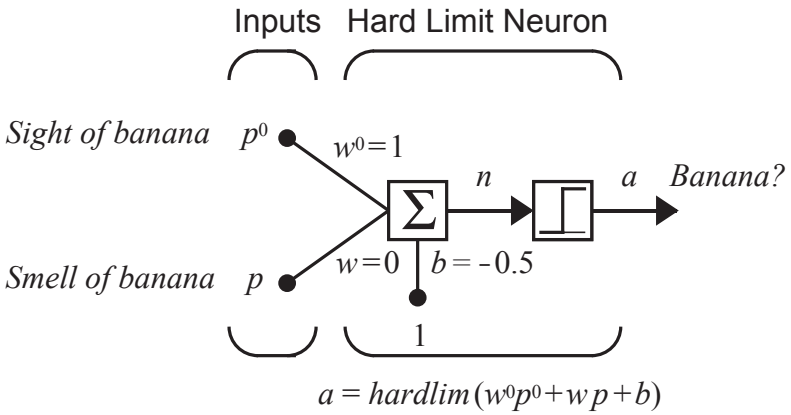
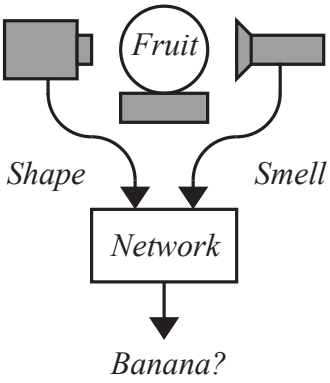


Figure 15.2 Banana Associator



The definitions of the unconditioned and conditioned inputs for this network are

$$p^0 = \begin{cases} 1, & \text{shape detected} \\ 0, & \text{shape not detected} \end{cases} \qquad p = \begin{cases} 1, & \text{smell detected} \\ 0, & \text{smell not detected} \end{cases} . \tag{15.3}$$

At this time we would like the network to associate the shape of a banana, but the not the smell, with a response indicating the fruit is a banana. The problem is solved by assigning a value greater than $-b$ to w^0 and assigning a value less than $-b$ to w. The following values satisfy these requirements:

$$w^0 = 1, \; w = 0. \tag{15.4}$$

The banana associator's input/output function now simplifies to

$$a = hardlim(p^0 - 0.5) . \tag{15.5}$$

Thus, the network will only respond if a banana is sighted ($p^0 = 1$), whether a banana is smelled ($p = 1$) or not ($p = 0$).

We will use this network in later sections to illustrate the performance of several associative learning rules.

Unsupervised Hebb Rule

For simple problems it is not difficult to design a network with a fixed set of associations. On the other hand, a more useful network would be able to learn associations.

When should an association be learned? It is generally accepted that both animals and humans tend to associate things that occur simultaneously. To paraphrase Hebb: if a banana smell stimulus occurs simultaneously with a banana concept response (activated by some other stimulus such as the sight of a banana shape), the network should strengthen the connection between them so that later it can activate its banana concept in response to the banana smell alone.

The unsupervised Hebb rule does just that by increasing the weight w_{ij} between a neuron's input p_j and output a_i in proportion to their product:

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) p_j(q) . \tag{15.6}$$

(See also Eq. (7.5).) The learning rate α dictates how many times a stimulus and response must occur together before an association is made. In the network in Figure 15.2, an association will be made when $w > -b = 0.5$, since then $p = 1$ will produce the response $a = 1$, regardless of the value of p^0.

Local Learning

Note that Eq. (15.6) uses only signals available within the layer containing the weights being updated. Rules that satisfy this condition are called *local learning* rules. This is in contrast to the backpropagation rule, for example, in which the sensitivity must be propagated back from the final layer. The rules introduced in this chapter will all be local learning rules.

The unsupervised Hebb rule can also be written in vector form:

$$\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha \mathbf{a}(q) \mathbf{p}^T(q) . \tag{15.7}$$

Training Sequence

As with all unsupervised rules, learning is performed in response to a series of inputs presented in time (the *training sequence*):

$$\mathbf{p}(1), \mathbf{p}(2), \ldots, \mathbf{p}(Q) . \tag{15.8}$$

(Note that we are using the notation $\mathbf{p}(q)$, instead of $\mathbf{p}_q$, in order to emphasize the time-sequence nature of the inputs.) At each iteration, the output $\mathbf{a}$ is calculated in response to the input $\mathbf{p}$, and then the weights $\mathbf{W}$ are updated with the Hebb rule.

Let's apply the unsupervised Hebb rule to the banana associator. The associator will start with the weight values determined in our previous example, so that it will initially respond to the sight, but not the smell, of a banana.

$$w^0 = 1, w(0) = 0 \tag{15.9}$$

The associator will be repeatedly exposed to a banana. However, while the network's smell sensor will work reliably, the shape sensor will operate only intermittently (on even time steps). Thus the training sequence will consist of repetitions of the following two sets of inputs:

$$\{p^0(1) = 0, p(1) = 1\}, \{p^0(2) = 1, p(2) = 1\}, \ldots . \tag{15.10}$$

The first weight w^0, representing the weight for the unconditioned stimulus p^0, will remain constant, while w will be updated at each iteration, using the unsupervised Hebb rule with a learning rate of 1:

$$w(q) = w(q-1) + a(q)p(q) . \tag{15.11}$$

The output for the first iteration ($q = 1$) is

$$\begin{aligned} a(1) &= hardlim(w^0 p^0(1) + w(0)p(1) - 0.5) \\ &= hardlim(1 \cdot 0 + 0 \cdot 1 - 0.5) = 0 \quad \text{(no response)} . \end{aligned} \tag{15.12}$$

The smell alone did not generate a response. Without a response, the Hebb rule does not alter w.

$$w(1) = w(0) + a(1)p(1) = 0 + 0 \cdot 1 = 0 \tag{15.13}$$

In the second iteration, both the banana's shape and smell are detected and the network responds accordingly:

$$\begin{aligned} a(2) &= hardlim(w^0 p^0(2) + w(1)p(2) - 0.5) \\ &= hardlim(1 \cdot 1 + 0 \cdot 1 - 0.5) = 1 \quad \text{(banana)} . \end{aligned} \tag{15.14}$$

Because the smell stimulus and the response have occurred simultaneously, the Hebb rule increases the weight between them.

$$w(2) = w(1) + a(2)p(2) = 0 + 1 \cdot 1 = 1 \tag{15.15}$$

When the sight detector fails again, in the third iteration, the network responds anyway. It has made a useful association between the smell of a banana and its response.

$$a(3) = hardlim(w^0 p^0(3) + w(2)p(3) - 0.5) \quad (15.16)$$
$$= hardlim(1 \cdot 0 + 1 \cdot 1 - 0.5) = 1 \quad \text{(banana)}$$

$$w(3) = w(2) + a(3)p(3) = 1 + 1 \cdot 1 = 2 \quad (15.17)$$

From now on, the network is capable of responding to bananas that are detected either by sight or smell. Even if both detection systems suffer intermittent faults, the network will be correct most of the time.

To experiment with the unsupervised Hebb rule, use the Neural Network Design Demonstration Unsupervised Hebb Rule (nnd13uh).

We have seen that the unsupervised Hebb rule can learn useful associations. However, the Hebb rule, as defined in Eq. (15.6), has some practical shortcomings. The first problem becomes evident if we continue to present inputs and update w in the example above. The weight w will become arbitrarily large. This is at odds with the biological systems that inspired the Hebb rule. Synapses cannot grow without bound.

The second problem is that there is no mechanism for weights to decrease. If the inputs or outputs of a Hebb network experience any noise, every weight will grow (however slowly) until the network responds to any stimulus.

Hebb Rule with Decay

One way to improve the Hebb rule is by adding a weight decay term (Eq. (7.45)),

$$\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha \mathbf{a}(q)\mathbf{p}^T(q) - \gamma \mathbf{W}(q-1)$$
$$= (1-\gamma)\mathbf{W}(q-1) + \alpha \mathbf{a}(q)\mathbf{p}^T(q)\,, \quad (15.18)$$

Decay Rate

where γ, the *decay rate*, is a positive constant less than one. As γ approaches zero, the learning law becomes the standard rule. As γ approaches one, the learning law quickly forgets old inputs and remembers only the most recent patterns. This keeps the weight matrix from growing without bound. (The idea of filtering the weight changes was also discussed in Chapter 12, where we called it momentum.)

The maximum weight value w_{ij}^{max} is determined by γ. This value is found by setting both a_i and p_j to a value of 1 for all q (to maximize learning) in the scalar version of Eq. (15.18) and solving for the steady state weight (i.e. when both new and old weights are equal).

$$w_{ij} = (1-\gamma)w_{ij} + \alpha a_i p_j$$
$$w_{ij} = (1-\gamma)w_{ij} + \alpha \qquad (15.19)$$
$$w_{ij} = \frac{\alpha}{\gamma}$$

Let's examine the operation of the Hebb rule with decay on our previous banana associator problem. We will use a decay rate γ of 0.1. The first iteration, where only the smell stimulus is presented, is the same:

$$a(1) = 0 \quad \text{(no response)}, \quad w(1) = 0 . \qquad (15.20)$$

The next iteration also produces identical results. Here both stimuli are presented, and the network responds to the shape. Coincidence of the smell stimulus and response create a new association:

$$a(2) = 1 \quad \text{(banana)}, \quad w(2) = 1 . \qquad (15.21)$$

The results of the third iteration are not the same. The network has learned to respond to the smell, and the weight continues to increase. However, this time the weight increases by only 0.9, instead of 1.0.

$$w(3) = w(2) + a(3)p(3) - 0.1w(2) = 1 + 1 \cdot 1 - 0.1 \cdot 1 = 1.9 \qquad (15.22)$$

The decay term limits the weight's value, so that no matter how often the association is reinforced, w will never increase beyond w_{ij}^{max}.

$$w_{ij}^{max} = \frac{\alpha}{\gamma} = \frac{1}{0.1} = 10 \qquad (15.23)$$

The new rule also ensures that associations learned by the network will not be artifacts of noise. Any small random increases will soon decay away.

Figure 15.3 displays the response of the Hebb rule, with and without decay, for the banana recognition example. Without decay, the weight continues to increase by the same amount each time the neuron is activated. When decay is added, the weight exponentially approaches its maximum value ($w_{ij}^{max} = 10$).

*To experiment with the Hebb rule with decay, use the Neural Network Design Demonstrations Hebb with Decay(*nnd13hd*) and Effect of Decay Rate* (nnd13edr).

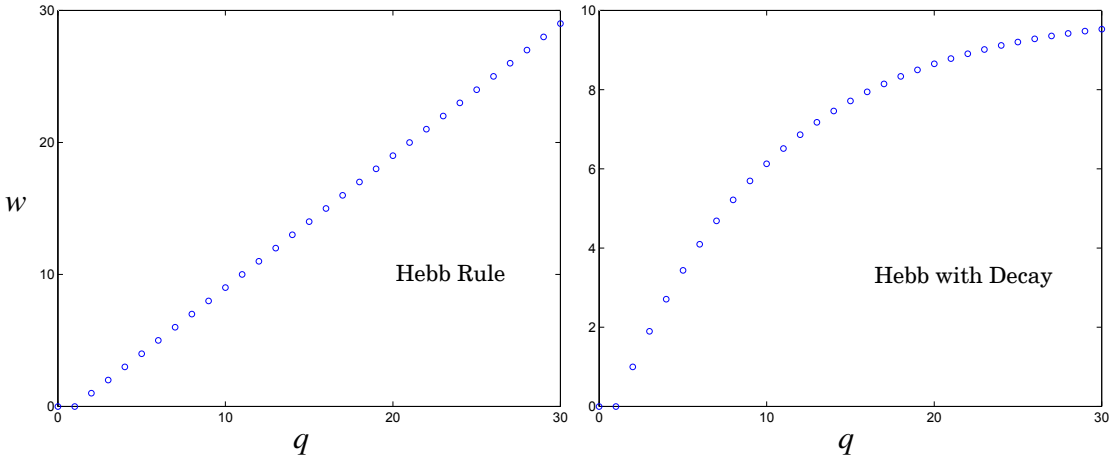


Figure 15.3 Response of the Hebb Rule, With and Without Decay

The Hebb rule with decay does solve the problem of large weights. However, it does so at a price. The environment must be counted on to occasionally present all stimuli that have associations. Without reinforcement, associations will decay away.

To illustrate this fact, consider Eq. (15.18) if $a_i = 0$:

$$w_{ij}(q) = (1-\gamma)w_{ij}(q-1). \tag{15.24}$$

If $\gamma = 0.1$, this reduces to

$$w_{ij}(q) = (0.9)w_{ij}(q-1). \tag{15.25}$$

Therefore w_{ij} will be decreased by 10% at each presentation for which $a_i = 0$. Any association that was previously learned will eventually be lost. We will discuss a solution to this problem in a later section.

Simple Recognition Network

Instar

So far we have considered only associations between scalar inputs and outputs. We will now examine a neuron that has a vector input. (See Figure 15.4.) This neuron, which is sometimes referred to as an *instar*, is the simplest network that is capable of pattern recognition, as we will demonstrate shortly.

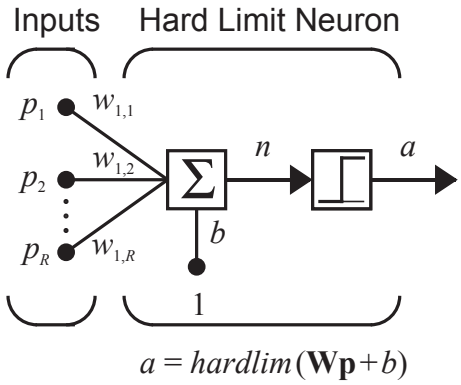


Figure 15.4 Instar

You will notice the similarities between the instar of Figure 15.4 and the perceptron of Figure 4.2 (also the ADALINE of Figure 10.2 and the linear associator of Figure 7.1). We give these networks different names, in part for historical reasons (since they arose at different times and out of different environments), and because they perform different functions and are analyzed in different ways. For example, we will not directly consider the decision boundary of the instar, although this was an important concept for the perceptron. Instead, we will analyze the ability of the instar to recognize a pattern, as with the neurons in the first layer of the Hamming network. (See page 3-10.)

The input/output expression for the instar is

$$a = hardlim(\mathbf{W}\mathbf{p} + b) = hardlim({}_1\mathbf{w}^T\mathbf{p} + b). \tag{15.26}$$

The instar will be active whenever the inner product between the weight vector (row of the weight matrix) and the input is greater than or equal to $-b$:

$${}_1\mathbf{w}^T\mathbf{p} \geq -b. \tag{15.27}$$

From our discussion of the Hamming network on page 3-10, we know that for two vectors of constant length, the inner product will be largest when they point in the same direction. We can also show this using Eq. (5.15):

$${}_1\mathbf{w}^T\mathbf{p} = \|{}_1\mathbf{w}\|\|\mathbf{p}\|\cos\theta \geq -b, \tag{15.28}$$

where θ is the angle between the two vectors. Clearly the inner product is maximized when the angle θ is 0. If $\mathbf{p}$ and ${}_1\mathbf{w}$ have the same length ($\|\mathbf{p}\| = \|{}_1\mathbf{w}\|$), then the inner product will be largest when $\mathbf{p} = {}_1\mathbf{w}$.

Based on these arguments, the instar of Figure 15.4 will be active when $\mathbf{p}$ is "close" to ${}_1\mathbf{w}$. By setting the bias b appropriately, we can select how close the input vector must be to the weight vector in order to activate the instar.

If we set

$$b = -\|{}_1\mathbf{w}\|\|\mathbf{p}\|, \tag{15.29}$$

then the instar will only be active when $\mathbf{p}$ points in exactly the same direction as ${}_1\mathbf{w}$ ($\theta = 0$). Thus, we will have a neuron that recognizes only the pattern ${}_1\mathbf{w}$.

If we would like the instar to respond to any pattern near ${}_1\mathbf{w}$ (θ small), then we can increase b to some value larger than $-\|{}_1\mathbf{w}\|\|\mathbf{p}\|$. The larger the value of b, the more patterns there will be that can activate the instar, thus making it the less discriminatory.

We should note that this analysis assumes that all input vectors have the same length (norm). We will revisit the question of normalization in Chapters 16, 18 and 19.

We can now design a vector recognition network if we know which vector we want to recognize. However, if the network is to learn a vector without supervision, we need a new rule, since neither version of the Hebb rule produces normalized weights.

Instar Rule

One problem of the Hebb rule with decay was that it required stimuli to be repeated or associations would be lost. A better rule might allow weight decay only when the instar is active ($a \neq 0$). Weight values would still be limited, but forgetting would be minimized. Consider again the original Hebb rule:

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q)p_j(q). \tag{15.30}$$

To get the benefits of weight decay, while limiting the forgetting problem, a decay term can be added that is proportional to $a_i(q)$:

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q)p_j(q) - \gamma a_i(q)w_{ij}(q-1) \tag{15.31}$$

We can simplify Eq. (15.31) by setting γ equal to α (so new weight values are learned at the same rate old values decay) and gathering terms.

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q)(p_j(q) - w_{ij}(q-1)) \tag{15.32}$$

Instar Rule

This equation, called the *instar rule*, can also be rewritten in vector form:

$${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha a_i(q)(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)). \tag{15.33}$$

The performance of the instar rule can be best understood if we consider the case where the instar is active ($a_i = 1$). Eq. (15.33) can then be written

$$
\begin{aligned}
{}_i\mathbf{w}(q) &= {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)) \\
&= (1-\alpha){}_i\mathbf{w}(q-1) + \alpha\mathbf{p}(q)\,.
\end{aligned} \tag{15.34}
$$

This operation is displayed graphically in Figure 15.5.

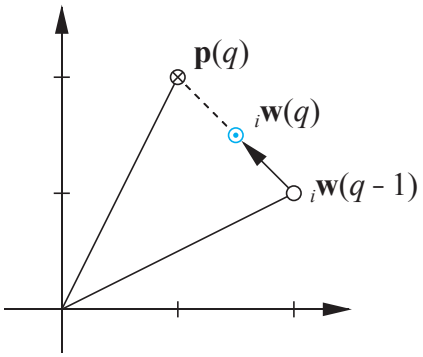
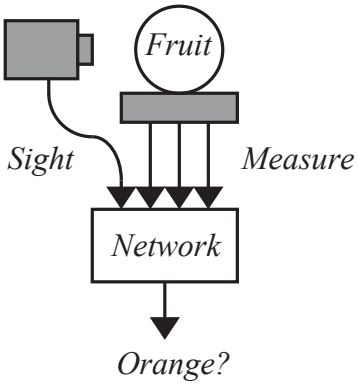


Figure 15.5 Graphical Representation of the Instar Rule

When the instar is active, the weight vector is moved toward the input vector along a line between the old weight vector and the input vector. The distance the weight vector moves depends on the value of α. When $\alpha = 0$, the new weight vector is equal to the old weight vector (no movement). When $\alpha = 1$, the new weight vector is equal to the input vector (maximum movement). If $\alpha = 0.5$, the new weight vector will be halfway between the old weight vector and the input vector.

One useful feature of the instar rule is that if the input vectors are normalized, then ${}_i\mathbf{w}$ will also be normalized once it has learned a particular vector $\mathbf{p}$. We have found a rule that not only minimizes forgetting, but results in normalized weight vectors, if the input vectors are normalized.

Let's apply the instar rule to the network in Figure 15.6. It has two inputs: one indicating whether a fruit has been visually identified as an orange (unconditioned stimulus) and another consisting of the three measurements taken of the fruit (conditioned stimulus).



The output of this network is

$$
a = hardlim(w^0 p^0 + \mathbf{W}\mathbf{p} + b)\,. \tag{15.35}
$$

The elements of input $\mathbf{p}$ will be constrained to ±1 values, as defined in Chapter 3 (Eq. (3.2)). This constraint ensures that $\mathbf{p}$ is a normalized vector with a length of $\|\mathbf{p}\| = \sqrt{3}$. The definitions of p^0 and $\mathbf{p}$ are

$$
p^0 = \begin{cases} 1, & \text{orange detected visually} \\ 0, & \text{orange not detected} \end{cases} \qquad \mathbf{p} = \begin{bmatrix} shape \\ texture \\ weight \end{bmatrix}. \tag{15.36}
$$

The bias b is -2, a value slightly more positive than $-\|\mathbf{p}\|^2 = -3$. (See Eq. (15.29).)

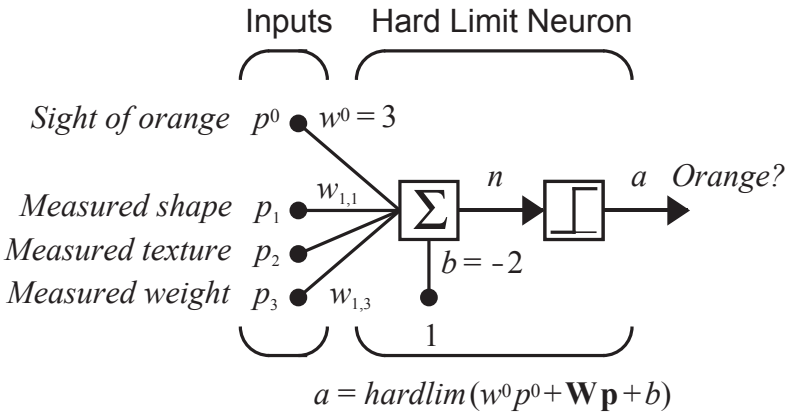


Figure 15.6 Orange Recognizer

We would like the network to have a constant association between the sight of an orange and its response, so w^0 will be set greater than $-b$. But initially, the network should not respond to any combination of fruit measurements, so the measurement weights will start with values of 0.

$$w^0 = 3, \quad \mathbf{W}(0) = {}_1\mathbf{w}^T(0) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \tag{15.37}$$

The measurement weights will be updated with the instar rule, using a learning rate of $\alpha = 1$.

$${}_1\mathbf{w}(q) = {}_1\mathbf{w}(q-1) + a(q)(\mathbf{p}(q) - {}_1\mathbf{w}(q-1)) \tag{15.38}$$

The training sequence will consist of repeated presentations of an orange. The measurements will be given every time. However, in order to demonstrate the operation of the instar rule, we will assume that the visual system only operates correctly on even time steps, due to a fault in its construction.

$$\left\{ p^0(1) = 0, \mathbf{p}(1) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right\}, \left\{ p^0(2) = 1, \mathbf{p}(2) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right\}, \ldots \tag{15.39}$$

Because $\mathbf{W}$ initially contains all zeros, the instar does not respond to the measurements of an orange in the first iteration.

$$a(1) = hardlim(w^0 p^0(1) + \mathbf{W}\mathbf{p}(1) - 2)$$

$$a(1) = hardlim\left(3 \cdot 0 + \begin{bmatrix}0 & 0 & 0\end{bmatrix}\begin{bmatrix}1\\-1\\-1\end{bmatrix} - 2\right) = 0 \quad \text{(no response)} \tag{15.40}$$

Since the neuron did not respond, its weights ${}_1\mathbf{w}$ are not altered by the instar rule.

$${}_1\mathbf{w}(1) = {}_1\mathbf{w}(0) + a(1)(\mathbf{p}(1) - {}_1\mathbf{w}(0)) \tag{15.41}$$

$$= \begin{bmatrix}0\\0\\0\end{bmatrix} + 0\left(\begin{bmatrix}1\\-1\\-1\end{bmatrix} - \begin{bmatrix}0\\0\\0\end{bmatrix}\right) = \begin{bmatrix}0\\0\\0\end{bmatrix}$$

However, the neuron does respond when the orange is identified visually, in addition to being measured, in the second iteration.

$$a(2) = hardlim(w^0 p^0(2) + \mathbf{W}\mathbf{p}(2) - 2) \tag{15.42}$$

$$= hardlim\left(3 \cdot 1 + \begin{bmatrix}0 & 0 & 0\end{bmatrix}\begin{bmatrix}1\\-1\\-1\end{bmatrix} - 2\right) = 1 \quad \text{(orange)}$$

The result is that the neuron learns to associate the orange's measurement vector with its response. The weight vector ${}_1\mathbf{w}$ becomes a copy of the orange measurement vector.

$${}_1\mathbf{w}(2) = {}_1\mathbf{w}(1) + a(2)(\mathbf{p}(2) - {}_1\mathbf{w}(1)) \tag{15.43}$$

$$= \begin{bmatrix}0\\0\\0\end{bmatrix} + 1\left(\begin{bmatrix}1\\-1\\-1\end{bmatrix} - \begin{bmatrix}0\\0\\0\end{bmatrix}\right) = \begin{bmatrix}1\\-1\\-1\end{bmatrix}$$

The network can now recognize the orange by its measurements. The neuron responds in the third iteration, even though the visual detection system failed again.

$$a(3) = hardlim(w^0 p^0(3) + \mathbf{W}\mathbf{p}(3) - 2)$$

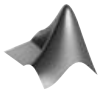
$$a(3) = hardlim\left(3 \cdot 0 + \begin{bmatrix}1 & -1 & -1\end{bmatrix}\begin{bmatrix}1\\-1\\-1\end{bmatrix} - 2\right) = 1 \quad \text{(orange)} \tag{15.44}$$

Having completely learned the measurements, the weights stop changing. (A lower learning rate would have required more iterations.)

$${}_1\mathbf{w}(3) = {}_1\mathbf{w}(2) + a(3)(\mathbf{p}(3) - {}_1\mathbf{w}(2)) \tag{15.45}$$

$$= \begin{bmatrix}1\\-1\\-1\end{bmatrix} + 1\left(\begin{bmatrix}1\\-1\\-1\end{bmatrix} - \begin{bmatrix}1\\-1\\-1\end{bmatrix}\right) = \begin{bmatrix}1\\-1\\-1\end{bmatrix}$$

The network has learned to recognize an orange by its measurements, even when its visual detection system fails.



To experiment with the instar rule, use the Neural Network Design Demonstrations Instar (`nnd13is`) *and Graphical Instar* (`nnd13gis`).

Kohonen Rule

Kohonen Rule

At this point it is appropriate to introduce another associative learning rule, which is related to the instar rule. It is the *Kohonen rule*:

$${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)), \quad \text{for } i \in X(q). \tag{15.46}$$

Like the instar rule, the Kohonen rule allows the weights of a neuron to learn an input vector and is therefore suitable for recognition applications. Unlike the instar rule, learning is not proportional to the neuron's output $a_i(q)$. Instead, learning occurs when the neuron's index i is a member of the set $X(q)$.

If the instar rule is applied to a layer of neurons whose transfer function only returns values of 0 or 1 (such as *hardlim*), then the Kohonen rule can be made equivalent to the instar rule by defining $X(q)$ as the set of all i such that $a_i(q) = 1$. The advantage of the Kohonen rule is that it can also be used with other definitions. It is useful for training networks such as the self-organizing feature map, which will be introduced in Chapter 16.

Simple Recall Network

Outstar

We have seen that the instar network (with a vector input and a scalar output) can perform pattern recognition by associating a particular vector stimulus with a response. The *outstar* network, shown in Figure 15.7, has a scalar input and a vector output. It can perform *pattern recall* by associating a stimulus with a vector response.

The input-output expression for this network is

$$\mathbf{a} = \mathbf{satlins}(\mathbf{W}p). \tag{15.47}$$

The symmetric saturating function **satlins** was chosen because this network will be used to recall a vector containing values of -1 or 1.

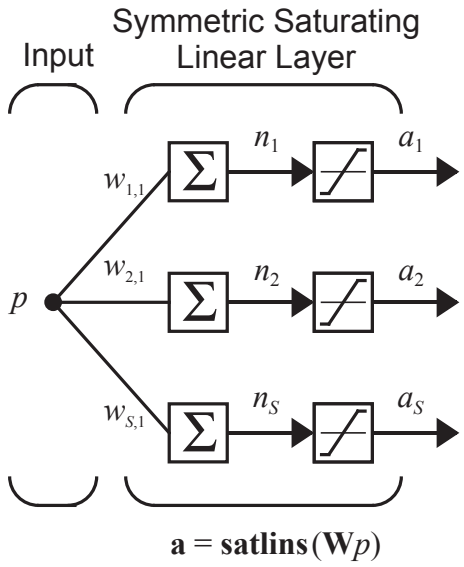


Figure 15.7 Outstar Network

If we would like the network to associate a stimulus (an input of 1) with a particular output vector $\mathbf{a}^*$, we can simply set $\mathbf{W}$ (which contains only a single column vector) equal to $\mathbf{a}^*$. Then, if p is 1, the output will be $\mathbf{a}^*$:

$$\mathbf{a} = \mathbf{satlins}(\mathbf{W}p) = \mathbf{satlins}(\mathbf{a}^* \cdot 1) = \mathbf{a}^*. \tag{15.48}$$

(This assumes that the elements of $\mathbf{a}^*$ are less than or equal to 1 in magnitude.)

Note that we have created a recall network by setting a *column* of the weight matrix to the desired vector. Earlier we designed a recognition network by setting a *row* of the weight matrix to the desired vector.

We can now design a network that can recall a known vector $\mathbf{a}^*$, but we need a learning rule if the network is to learn a vector without supervision. We will describe such a learning rule in the next section.

Outstar Rule

To derive the instar rule, forgetting was limited by making the weight decay term of the Hebb rule proportional to the output of the network, a_i. Conversely, to obtain the outstar learning rule, we make the weight decay term proportional to the input of the network, p_j:

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q)p_j(q) - \gamma p_j(q)w_{ij}(q-1). \tag{15.49}$$

If we set the decay rate γ equal to the learning rate α and collect terms, we get

$$w_{ij}(q) = w_{ij}(q-1) + \alpha(a_i(q) - w_{ij}(q-1))p_j(q). \tag{15.50}$$

The outstar rule has properties complimentary to the instar rule. Learning occurs whenever p_j is nonzero (instead of a_i). When learning occurs, column $\mathbf{w}_j$ moves toward the output vector.

Outstar Rule

As with the instar rule, the *outstar rule* can be written in vector form:

$$\mathbf{w}_j(q) = \mathbf{w}_j(q-1) + \alpha(\mathbf{a}(q) - \mathbf{w}_j(q-1))p_j(q), \tag{15.51}$$

where $\mathbf{w}_j$ is the jth column of the matrix $\mathbf{W}$.

To test the outstar rule we will train the network shown in Figure 15.8.

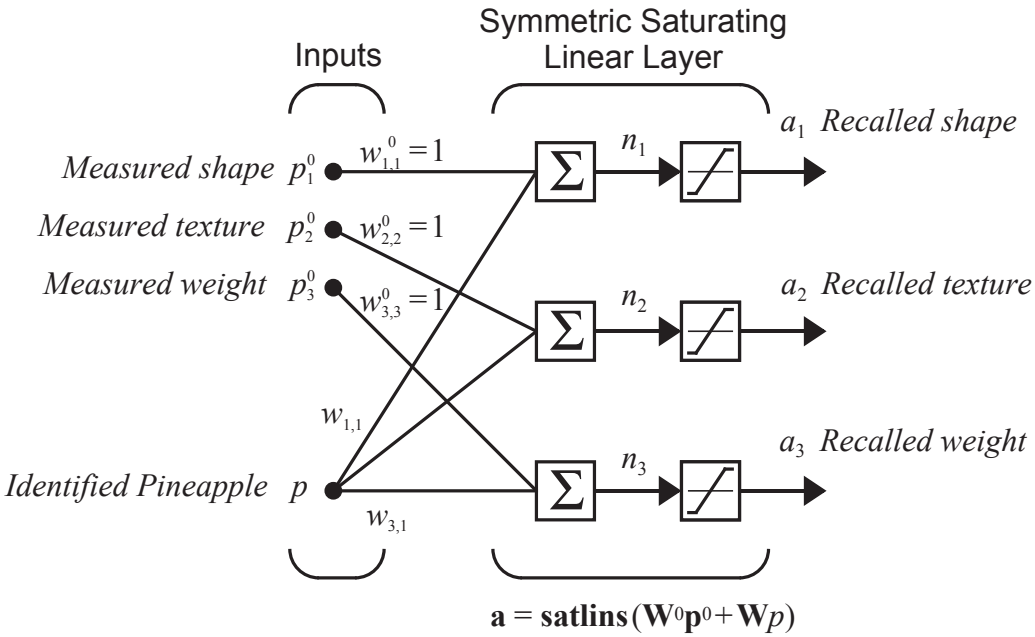
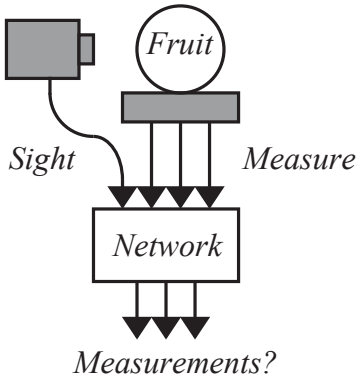


Figure 15.8 Pineapple Recaller

The outputs of the network are calculated as follows:

$$\mathbf{a} = \mathbf{satlins}(\mathbf{W}^0\mathbf{p}^0 + \mathbf{W}p), \tag{15.52}$$

where



$$\mathbf{W}^0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{15.53}$$

The network's two inputs provide it with measurements $\mathbf{p}^0$ taken on a fruit (unconditioned stimulus), as well as a signal p indicating a pineapple has been identified visually (conditioned stimulus).

$$\mathbf{p}^0 = \begin{bmatrix} shape \\ texture \\ weight \end{bmatrix} \qquad p = \begin{cases} 1, & \text{if a pineapple can be seen} \\ 0, & \text{otherwise} \end{cases} \tag{15.54}$$

The network's output is to reflect the measurements of the fruit currently being examined, using whatever inputs are available.

The weight matrix for the unconditioned stimulus, $\mathbf{W}^0$, is set to the identity matrix, so that any set of measurements $\mathbf{p}^0$ (with ±1 values) will be copied to the output $\mathbf{a}$. The weight matrix for the conditioned stimulus, $\mathbf{W}$, is set to zero initially, so that a 1 on p will not generate a response. $\mathbf{W}$ will be updated with the outstar rule using a learning rate of 1:

$$\mathbf{w}_j(q) = \mathbf{w}_j(q-1) + (\mathbf{a}(q) - \mathbf{w}_j(q-1))p(q). \tag{15.55}$$

The training sequence will consist of repeated presentations of the sight and measurements of a pineapple. The pineapple measurements are

$$\mathbf{p}^{pineapple} = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}. \tag{15.56}$$

However, due to a fault in the measuring system, measured values will only be available on even iterations.

$$\left\{\mathbf{p}^0(1) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, p(1) = 1\right\}, \left\{\mathbf{p}^0(2) = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, p(2) = 1\right\}, \ldots \tag{15.57}$$

In the first iteration, the pineapple is seen, but the measurements are unavailable.

$$\mathbf{a}(1) = \mathbf{satlins}(\mathbf{W}^0\mathbf{p}^0(1) + \mathbf{W}p(1)), \tag{15.58}$$

$$\mathbf{a}(1) = \mathbf{satlins}\left(\begin{bmatrix}0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\end{bmatrix}1\right) = \begin{bmatrix}0\\0\\0\end{bmatrix} \quad \text{(no response)} \tag{15.59}$$

The network sees the pineapple but cannot output proper measurements, because it has not learned them and the measurement system is not working. The weights remain unchanged after being updated.

$$\mathbf{w}_1(1) = \mathbf{w}_1(0) + (\mathbf{a}(1) - \mathbf{w}_1(0))p(1) = \begin{bmatrix}0\\0\\0\end{bmatrix} + \left(\begin{bmatrix}0\\0\\0\end{bmatrix} - \begin{bmatrix}0\\0\\0\end{bmatrix}\right)1 = \begin{bmatrix}0\\0\\0\end{bmatrix} \tag{15.60}$$

In the second iteration the pineapple is seen, and the measurements are taken properly.

$$\mathbf{a}(2) = \mathbf{satlins}\left(\begin{bmatrix}-1\\-1\\1\end{bmatrix} + \begin{bmatrix}0\\0\\0\end{bmatrix}1\right) = \begin{bmatrix}-1\\-1\\1\end{bmatrix} \quad \text{(measurements given)} \tag{15.61}$$

The measurements are available, so the network outputs them correctly. The weights are then updated as follows:

$$\begin{aligned}\mathbf{w}_1(2) &= \mathbf{w}_1(1) + (\mathbf{a}(2) - \mathbf{w}_1(1))p(2)\\ &= \begin{bmatrix}0\\0\\0\end{bmatrix} + \left(\begin{bmatrix}-1\\-1\\1\end{bmatrix} - \begin{bmatrix}0\\0\\0\end{bmatrix}\right)1 = \begin{bmatrix}-1\\-1\\1\end{bmatrix}.\end{aligned} \tag{15.62}$$

Since the sight of the pineapple and the measurements were both available, the network forms an association between them. The weight matrix is now a copy of the measurements, so they can be recalled later.

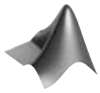
In iteration three, measurements are unavailable once again, but the output is

$$\mathbf{a}(3) = \mathbf{satlins}\left(\begin{bmatrix}0\\0\\0\end{bmatrix} + \begin{bmatrix}-1\\-1\\1\end{bmatrix}1\right) = \begin{bmatrix}-1\\-1\\1\end{bmatrix} \quad \text{(measurements recalled)}. \quad (15.63)$$

The network is now able to recall the measurements of the pineapple when it sees it, even though the measurement system fails. From now on, the weights will no longer change values unless a pineapple is seen with different measurements.

$$\mathbf{w}_1(3) = \mathbf{w}_1(2) + (\mathbf{a}(2) - \mathbf{w}_1(2))p(2)$$

$$= \begin{bmatrix}-1\\-1\\1\end{bmatrix} + \left(\begin{bmatrix}-1\\-1\\1\end{bmatrix} - \begin{bmatrix}-1\\-1\\1\end{bmatrix}\right)1 = \begin{bmatrix}-1\\-1\\1\end{bmatrix} \quad (15.64)$$



*To experiment with the outstar rule with decay, use the Neural Network Design Demonstration Outstar Rule (*nnd13os*).*

In Chapter 19 we will investigate the ART networks, which use both the instar and the outstar rules.

Summary of Results

Association

An association is a link between the inputs and outputs of a network so that when a stimulus A is presented to the network, it will output a response B.

Associative Learning Rules

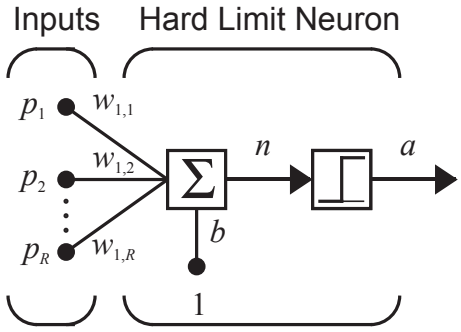
Unsupervised Hebb Rule

$$\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha \mathbf{a}(q)\mathbf{p}^T(q)$$

Hebb Rule with Decay

$$\mathbf{W}(q) = (1-\gamma)\mathbf{W}(q-1) + \alpha \mathbf{a}(q)\mathbf{p}^T(q)$$

Instar



$$a = hardlim(\mathbf{W}\mathbf{p}+b)$$

$$a = hardlim({}_1\mathbf{w}^T\mathbf{p} + b)$$

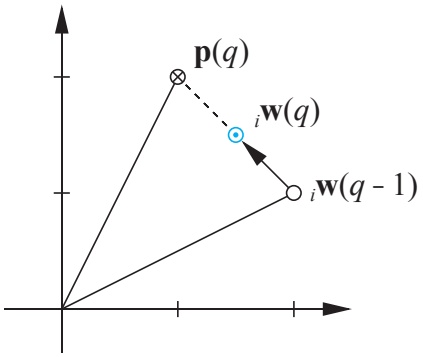
The instar is activated for ${}_1\mathbf{w}^T\mathbf{p} = \|{}_1\mathbf{w}\|\|\mathbf{p}\|\cos\theta \geq -b$,

where θ is the angle between $\mathbf{p}$ and ${}_1\mathbf{w}$.

Instar Rule

$${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha a_i(q)(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$$

$${}_i\mathbf{w}(q) = (1-\alpha)\mathbf{w}(q-1) + \alpha \mathbf{p}(q), \quad if(a_i(q) = 1)$$

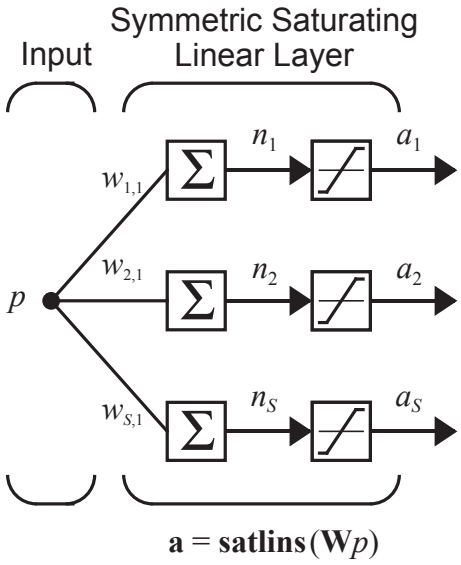


Graphical Representation of the Instar Rule ($a_i(q) = 1$)

Kohonen Rule

$$_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)), \quad \text{for } i \in X(q)$$

Outstar



Outstar Rule

$$\mathbf{w}_j(q) = \mathbf{w}_j(q-1) + \alpha(\mathbf{a}(q) - \mathbf{w}_j(q-1))p_j(q)$$

Solved Problems

P15.1 In Eq. (15.19) the maximum weight for the Hebb rule with decay was calculated, assuming that p_j and a_i were 1 at every time step. Calculate the maximum weight resulting if p_j and a_i alternate together between values of 0 and 1.

We begin with the scalar version of the Hebb rule with decay:

$$w_{ij}(q) = (1-\gamma)w_{ij}(q-1) + \alpha a_i(q)p_j(q).$$

We can rewrite this expression twice using q to index the weight values as the weight is updated over two time steps.

$$w_{ij}(q+1) = (1-\gamma)w_{ij}(q) + \alpha a_i(q)p_j(q)$$

$$w_{ij}(q+2) = (1-\gamma)w_{ij}(q+1) + \alpha a_i(q+1)p_j(q+1)$$

By substituting the first equation into the second, we get a single expression showing how w_{ij} is updated over two time steps.

$$w_{ij}(q+2) = (1-\gamma)((1-\gamma)w_{ij}(q) + \alpha a_i(q)p_j(q)) + \alpha a_i(q+1)p_j(q+1)$$

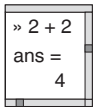
At this point we can substitute values for p_j and a_i. Because we are looking for a maximum weight, we will set $p_j(q)$ and $a_i(q)$ to 0, and $p_j(q+1)$ and $a_i(q+1)$ to 1. This will mean that the weight decreases in the first time step, and increases in the second, ensuring that $w_{ij}(q+2)$ is the maximum of the two weights. If we solve for $w_{ij}(q+2)$, we obtain

$$w_{ij}(q+2) = (1-\gamma)^2 w_{ij}(q) + \alpha.$$

Assuming that w_{ij} will eventually reach a steady state value, we can find it by setting both $w_{ij}(q+2)$ and $w_{ij}(q)$ equal to w_{ij}^{max} and solving

$$w_{ij}^{max} = (1-\gamma)^2 w_{ij}^{max} + \alpha,$$

$$w_{ij}^{max} = \frac{\alpha}{2\gamma - \gamma^2}.$$



We can use MATLAB to make a plot of this relationship. The plot will show learning rates and decay rates at intervals of 0.025.

```
lr = 0:0.025:1;
dr = 0.025:0.025:1;
```

Here are the commands for creating a mesh plot of the maximum weight, as a function of the learning and decay rate values.

```
[LR,DR] = meshgrid(dr,lr);
MW = LR ./ (DR .* (2 - DR));
mesh(DR,LR,MW);
```

The plot shows that w_{ij}^{max} approaches infinity as the decay rate γ becomes small with respect to the learning rate α (see Figure P15.1).

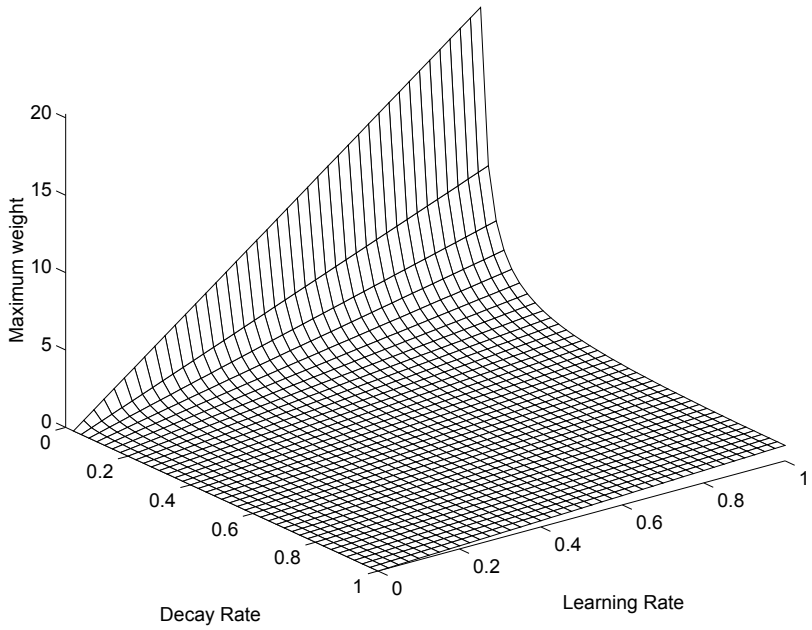


Figure P15.1 Maximum Weight w_{ij}^{max}

P15.2 Retrain the orange recognition network on page 13-13 using the instar rule with a learning rate of 0.4. Use the same training sequence. How many time steps are required for the network to learn to recognize an orange by its measurements?

Here is the training sequence. It is to be repeated until the network can respond to the orange measurements ($\mathbf{p} = \begin{bmatrix}1 & -1 & -1\end{bmatrix}^T$), even when the visual detection system fails ($p^0 = 0$).

$$\left\{p^0(1) = 0, \mathbf{p}(1) = \begin{bmatrix}1\\-1\\-1\end{bmatrix}\right\}, \left\{p^0(2) = 1, \mathbf{p}(2) = \begin{bmatrix}1\\-1\\-1\end{bmatrix}\right\}, \ldots$$

```
» 2 + 2
ans =
     4
```

We will use MATLAB to make the calculations. These two lines of code set the weights to their initial values.

```
w0 = 3;
W = [0 0 0];
```

We can then simulate the first time step of the network.

```
p0 = 0;
p = [1; -1; -1];
a = hardlim(w0*p0+W*p-2)
a =
     0
```

The neuron does not yet recognize the orange, so its output is 0. The weights do not change when updated with the instar rule.

```
W = W + 0.4*a*(p'-W)
W =
     0 0 0
```

The neuron begins learning the measurements in the second iteration.

```
p0 = 1;
p = [1; -1; -1];
a = hardlim(w0*p0+W*p-2)
a =
     1

W = W + 0.4*a*(p'-W)
W =
     0.4000 -0.4000 -0.4000
```

But the association is still not strong enough for a response in the third iteration.

```
p0 = 0;
p = [1; -1; -1];
a = hardlim(w0*p0+W*p-2)
a =
     0

W = W + 0.4*a*(p'-W)
W =
     0.4000 -0.4000 -0.4000
```

Here are the results of the fourth iteration:

```
a =
     1
W =
     0.6400 -0.6400 -0.6400
```

the fifth iteration:

```
a =
     0
W =
     0.6400 -0.6400 -0.6400
```

and the sixth iteration:

```
a =
     1
W =
     0.7840 -0.7840 -0.7840 .
```

By the seventh iteration the network is able to recognize the orange by its measurements alone.

```
p0 = 0;
p = [1; -1; -1];
a = hardlim(w0*p0+W*p-2)
a =
     1

W = W + 0.4*a*(p'-W)
W =
     0.8704 -0.8704 -0.8704
```

Due to the lower learning rate, the network had to experience the measurements paired with its response three times (the even numbered iterations) before it developed a strong association between them.

P15.3 Both the recognition and recall networks used in this chapter's examples could only learn a single vector. Draw the diagram and determine the parameters of a network capable of recognizing and responding to the following two vectors:

$$\mathbf{p}_1 = \begin{bmatrix} 5 \\ -5 \\ 5 \end{bmatrix} \qquad \mathbf{p}_2 = \begin{bmatrix} -5 \\ 5 \\ 5 \end{bmatrix}.$$

The network should only respond when an input vector is identical to one of these vectors.

We know the network must have three inputs, because it must recognize three-element vectors. We also know that it will have two outputs, one output for each response.

Such a network can be obtained by combining two instars into a single layer, as in Figure P15.2.

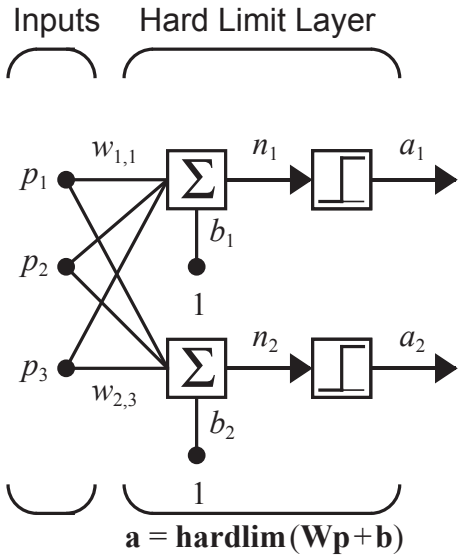


Figure P15.2 Two-Vector Recognition Network

We now set the weights ${}_1\mathbf{w}$ of the first neuron equal to $\mathbf{p}_1$, so that its net input will be at a maximum when an input vector points in the same direction as $\mathbf{p}_1$. Likewise, we will set ${}_2\mathbf{w}$ to $\mathbf{p}_2$ so that the second neuron is most sensitive to vectors in the direction of $\mathbf{p}_2$.

Combining the weight vectors gives us the weight matrix

$$\mathbf{W} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \end{bmatrix} = \begin{bmatrix} 5 & -5 & 5 \\ -5 & 5 & 5 \end{bmatrix}.$$

(Note that this is the same manner in which we determined the weight matrix for the first layer of the Hamming network. In fact, the first layer of the Hamming network is a layer of instars. More about that in the next chapter.)

The lengths of $\mathbf{p}_1$ and $\mathbf{p}_2$ are the same:

$$\|\mathbf{p}_1\| = \|\mathbf{p}_2\| = \sqrt{(5)^2 + (-5)^2 + (5)^2} = \sqrt{75}.$$

To ensure that only an exact match between an input vector and a stored vector results in a response, both biases are set as follows (Eq. (15.29)):

$$b_1 = b_2 = -\|\mathbf{p}_1\|^2 = -75.$$

```
» 2 + 2
ans =
     4
```

We can use MATLAB to test that the network does indeed respond to $\mathbf{p}_1$.

```
W = [5 -5 5; -5 5 5];
b = [-75; -75];
```

```
p1 = [5; -5; 5];
a = hardlim(W*p1+b)
a =
     1
     0
```

The first neuron responded, indicating that the input vector was $\mathbf{p}_1$. The second neuron did not respond, because the input was not $\mathbf{p}_2$.

We can also check that the network does not respond to a third vector $\mathbf{p}_3$ that is not equal to either of the stored vectors.

```
p3 = [-5; 5; -5];
a = hardlim(W*p3+b)
a =
     0

     0
```

Neither neuron recognizes this new vector, so they both output 0.

P15.4 A single instar is being used for pattern recognition. Its weights and bias have the following values:

$$\mathbf{W} = {}_1\mathbf{w}^T = \begin{bmatrix} 1 & -1 & -1 \end{bmatrix} \qquad b = -2 .$$

How close must an input vector (with a magnitude of $\sqrt{3}$) be to the weight vector for the neuron to output a 1? Find a vector that occurs on the border between those vectors that are recognized and those vectors that are not.

We begin by writing the expression for the neuron's output.

$$a = hardlim({}_1\mathbf{w}^T\mathbf{p} + b)$$

According to the definition of *hardlim*, a will be 1 if and only if the inner product between ${}_1\mathbf{w}^T$ and $\mathbf{p}$ is greater than or equal to $-b$ (Eq. (15.28)):

$${}_1\mathbf{w}^T\mathbf{p} = \|{}_1\mathbf{w}\|\,\|\mathbf{p}\|\cos\theta \geq -b .$$

We can find the maximum angle between ${}_1\mathbf{w}$ and $\mathbf{p}$ that meets this condition by substituting for the norms and solving

$$(\sqrt{3})(\sqrt{3})\cos\theta \geq 2$$

$$\theta \leq \cos^{-1}\left(\frac{2}{3}\right) = 48.19^\circ .$$

To find a borderline vector with magnitude $\sqrt{3}$, we need a vector $\mathbf{p}$ that meets the following conditions:

$$\|\mathbf{p}_1\| = \sqrt{p_1^2 + p_2^2 + p_3^2} = \sqrt{3},$$

$$_1\mathbf{w}^T\mathbf{p} = w_1p_1 + w_2p_2 + w_3p_3 - b = p_1 - p_2 - p_3 - 2 = 0.$$

Since we have three variables and only two constraints, we can set the third variable p_1 to 0 and solve

$$\sqrt{p_1^2 + p_2^2 + p_3^2} = \sqrt{3} \quad \Rightarrow \quad p_2^2 + p_3^2 = 3,$$

$$p_1 - p_2 - p_3 - 2 \quad \Rightarrow \quad p_2 + p_3 = -2,$$

$$(p_2 + p_3)^2 = p_2^2 + p_3^2 + 2p_2p_3 = (-2)^2 = 4,$$

$$3 + 2p_2p_3 = 4 \quad \Rightarrow \quad p_2p_3 = 0.5,$$

$$p_2(p_2 + p_3) = p_2^2 + p_2p_3 = p_2^2 + 0.5 = p_2(-2) = -2p_2.$$

After a little work we find that there are two possible values for p_2:

$$p_2^2 + 2p_2 + 0.5 = 0,$$

$$p_2 = -1 \pm \sqrt{0.5}.$$

It turns out that if we pick p_2 to be one of these values, then p_3 will take on the other value.

$$p_2 + p_3 = -1 \pm \sqrt{0.5} + p_3 = -2$$

$$p_3 = -1 \mp \sqrt{0.5}$$

Therefore, the following vector $\mathbf{p}$ is just the right distance from $\mathbf{w}$ to be recognized.

$$\mathbf{p} = \begin{bmatrix} 0 \\ -1 + \sqrt{0.5} \\ -1 - \sqrt{0.5} \end{bmatrix}$$

We can test it by presenting it to the network.

$$a = hardlim({}_1\mathbf{w}^T\mathbf{p} + b)$$

$$a = hardlim\left(\begin{bmatrix}1 & -1 & -1\end{bmatrix}\begin{bmatrix}0\\ -1+\sqrt{0.5}\\ -1-\sqrt{0.5}\end{bmatrix} - 2\right)$$

$$a = hardlim(0) = 1$$

The vector $\mathbf{p}$ does indeed result in a net input of 0 and is therefore on the boundary of the instar's active region.

P15.5 Consider the instar network shown in Figure P15.3. The training sequence for this network will consist of the following inputs:

$$\left\{p^0(1) = 0, \mathbf{p}(1) = \begin{bmatrix}-1\\ 1\end{bmatrix}\right\}, \left\{p^0(2) = 1, \mathbf{p}(2) = \begin{bmatrix}-1\\ 1\end{bmatrix}\right\}, \ldots .$$

These two sets of inputs are repeatedly presented to the network until the weight matrix $\mathbf{W}$ converges.

i. Perform the first four iterations of the instar rule, with learning rate $\alpha = 0.5$. Assume that the initial $\mathbf{W}$ matrix is set to all zeros.

ii. Display the results of each iteration of the instar rule in graphical form (as in Figure 15.5).

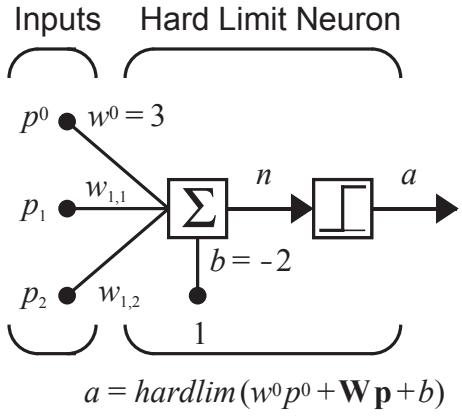


Figure P15.3 Instar Network for Problem P15.5

i. Because $\mathbf{W}$ initially contains all zeros, the instar does not respond to the measurements in the first iteration.

$$a(1) = hardlim(w^0 p^0(1) + \mathbf{W}\mathbf{p}(1) - 2)$$

$$a(1) = hardlim\left(3 \cdot 0 + \begin{bmatrix}0 & 0\end{bmatrix}\begin{bmatrix}-1\\1\end{bmatrix} - 2\right) = 0$$

The neuron did not respond. Therefore its weights ${}_1\mathbf{w}$ are not altered by the instar rule.

$$\begin{aligned} {}_1\mathbf{w}(1) &= {}_1\mathbf{w}(0) + 0.5a(1)(\mathbf{p}(1) - {}_1\mathbf{w}(0)) \\ &= \begin{bmatrix}0\\0\end{bmatrix} + 0\left(\begin{bmatrix}-1\\1\end{bmatrix} - \begin{bmatrix}0\\0\end{bmatrix}\right) = \begin{bmatrix}0\\0\end{bmatrix} \end{aligned}$$

Because the unconditioned stimulus appears on the second iteration, the instar does respond.

$$a(2) = hardlim(w^0 p^0(2) + \mathbf{W}\mathbf{p}(2) - 2)$$

$$a(2) = hardlim\left(3 \cdot 1 + \begin{bmatrix}0 & 0\end{bmatrix}\begin{bmatrix}-1\\1\end{bmatrix} - 2\right) = 1$$

The neuron did respond, and its weights ${}_1\mathbf{w}$ are updated by the instar rule.

$$\begin{aligned} {}_1\mathbf{w}(2) &= {}_1\mathbf{w}(1) + 0.5a(2)(\mathbf{p}(2) - {}_1\mathbf{w}(1)) \\ &= \begin{bmatrix}0\\0\end{bmatrix} + 0.5\left(\begin{bmatrix}-1\\1\end{bmatrix} - \begin{bmatrix}0\\0\end{bmatrix}\right) = \begin{bmatrix}-0.5\\0.5\end{bmatrix} \end{aligned}$$

On the third iteration, the unconditioned stimulus is not presented, and the weights have not yet converged close enough to the input pattern. Therefore, the instar does not respond.

$$a(3) = hardlim(w^0 p^0(3) + \mathbf{W}\mathbf{p}(3) - 2)$$

$$a(3) = hardlim\left(3 \cdot 0 + \begin{bmatrix}-0.5 & 0.5\end{bmatrix}\begin{bmatrix}-1\\1\end{bmatrix} - 2\right) = 0$$

Since the neuron did not respond, its weights are not updated.

$$
{}_1\mathbf{w}(3) = {}_1\mathbf{w}(2) + 0.5a(3)(\mathbf{p}(3) - {}_1\mathbf{w}(2))
$$

$$
= \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} + 0\left(\begin{bmatrix} -1 \\ 1 \end{bmatrix} - \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} \right) = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix}
$$

Because the unconditioned stimulus again appears on the fourth iteration, the instar does respond.

$$
a(4) = hardlim(w^0 p^0(4) + \mathbf{W}\mathbf{p}(4) - 2)
$$

$$
a(4) = hardlim\left(3 \cdot 1 + \begin{bmatrix} -0.5 & 0.5 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} - 2 \right) = 1
$$

Since the instar was activated, its weights are updated.

$$
{}_1\mathbf{w}(4) = {}_1\mathbf{w}(3) + 0.5a(4)(\mathbf{p}(4) - {}_1\mathbf{w}(3))
$$

$$
= \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} + 0.5\left(\begin{bmatrix} -1 \\ 1 \end{bmatrix} - \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} \right) = \begin{bmatrix} -0.75 \\ 0.75 \end{bmatrix}
$$

This completes the fourth iteration. If we continue this process, ${}_1\mathbf{w}$ will converge to $\mathbf{p}$.

ii. Note that the weights are only updated (instar active) on iterations 2 and 4. Recall from Eq. (15.34) that when the instar is active, the learning rule can be written

$$
{}_1\mathbf{w}(q) = {}_1\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_1\mathbf{w}(q-1)) = (1-\alpha){}_1\mathbf{w}(q-1) + \alpha\mathbf{p}(q).
$$

When the instar is active, the weight vector is moved toward the input vector along a line between the old weight vector and the input vector. Figure P15.4 displays the movement of the weight vector for this problem. The weights were updated on iterations 2 and 4. Because $\alpha = 0.5$, whenever the instar is active the weight vector moves halfway from its current position toward the input vector.

$$
{}_1\mathbf{w}(q) = (0.5){}_1\mathbf{w}(q-1) + (0.5)\mathbf{p}(q)
$$

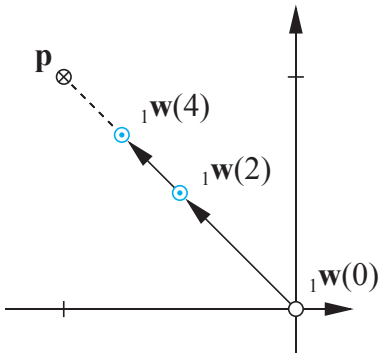


Figure P15.4 Instar Rule Example

Epilogue

In this chapter we introduced some simple networks capable of forming associations. We also developed and studied several learning rules that allowed networks to create new associations. Each rule operated by strengthening an association between any stimulus and response that occurred simultaneously.

The simple associative networks and learning rules developed in this chapter are useful in themselves, but they are also important building blocks for more powerful networks. In this chapter we introduced two networks, and associated learning rules, that will be fundamental for the development of important networks in the next three chapters: the instar and the outstar. The instar is a network that is trained to *recognize* a pattern. The outstar is a network that is trained to *recall* a pattern. We will use layers of instars in Chapters 16 and 18 to perform pattern recognition. These networks are very similar to the Hamming network of Chapter 3, whose first layer was, in fact, a layer of instars. In Chapter 19 we will introduce a more complex network, which combines both instars and outstars in order to produce stable learning.

Further Reading

[Ande72] J. Anderson, "A simple neural network generating an interactive memory," *Mathematical Biosciences*, vol. 14, pp. 197–220, 1972.

Anderson has proposed a "linear associator" model for associative memory. The model was trained, using a generalization of the Hebb postulate, to learn an association between input and output vectors. The physiological plausibility of the network was emphasized. Kohonen published a closely related paper at the same time [Koho72], although the two researchers were working independently.

[Gros68] S. Grossberg, "Some physiological and biochemical consequences of psychological postulates," *Proceedings of the National Academy of Sciences*, vol. 60, pp. 758–765, 1968.

This article describes early mathematical models (nonlinear differential equations) of associative learning. It synthesizes psychological, mathematical and physiological ideas.

[Gros82] S. Grossberg, *Studies of Mind and Brain*, Boston: D. Reidel Publishing Co., 1982.

This book is a collection of Stephen Grossberg papers from the period 1968 through 1980. It covers many of the fundamental concepts which are used in later Grossberg networks, such as the Adaptive Resonance Theory networks.

[Hebb49] D. O. Hebb, *The Organization of Behavior*, New York: Wiley, 1949.

The main premise of this seminal book was that behavior could be explained by the action of neurons. In it, Hebb proposed one of the first learning laws, which postulated a mechanism for learning at the cellular level.

[Koho72] T. Kohonen, "Correlation matrix memories," *IEEE Transactions on Computers*, vol. 21, pp. 353–359, 1972.

Kohonen proposed a correlation matrix model for associative memory. The model was trained, using the outer product rule (also known as the Hebb rule), to learn an association between input and output vectors. The mathematical structure of the network was emphasized. Anderson published a closely related paper at the same time [Ande72], although the two researchers were working independently.

[Koho87] T. Kohonen, *Self-Organization and Associative Memory*, 2nd Ed., Berlin: Springer-Verlag, 1987.

This book introduces the Kohonen rule and several networks that use it. It provides a complete analysis of linear associative models and gives many extensions and examples.

[Leib90] D. Lieberman, *Learning, Behavior and Cognition*, Belmont, CA: Wadsworth, 1990.

Leiberman's text forms an excellent introduction to behavioral psychology. This field is of interest to anyone looking to model human (or animal) learning with neural networks.

Exercises

E15.1 The network shown in Figure E15.1 is to be trained using the Hebb rule with decay, using a learning rate α of 0.3 and a decay rate γ of 0.1.

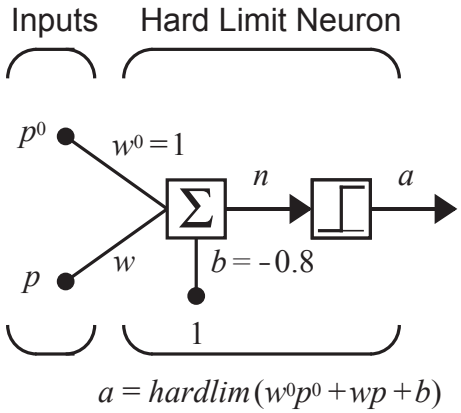


Figure E15.1 Associative Network

i. If w is initially set to 0, and w^0 and b remain constant (with the values shown in Figure E15.1), how many consecutive presentations of the following training set are required before the neuron will respond to the test set? Make a plot of w versus iteration number.

Training set: $\{p^0 = 1, p = 1\}$ Test set: $\{p^0 = 0, p = 1\}$

ii. Assume that w has an initial value of 1. How many consecutive presentations of the following training set are required before the neuron will no longer be able to respond to the test set? Make a plot of w versus iteration number.

Training set: $\{p^0 = 0, p = 0\}$ Test set: $\{p^0 = 0, p = 1\}$

E15.2 For Exercise E15.1 part (i), use Eq. (15.19) to determine the steady state value of w. Verify that this answer agrees with your plot from Exercise E15.1 part (i).

E15.3 Repeat Exercise E15.1, but this time use the Hebb rule without decay ($\gamma = 0$).

E15.4 The following rule looks similar to the instar rule, but it behaves quite differently:

$$\Delta w_{ij} = -\alpha a_i (p_j + w_{ij}^{old})$$

i. Determine the conditions under which the Δw_{ij} is nonzero.

ii. What value does the weight approach when Δw_{ij} is nonzero?

iii. Can you think of a use for this rule?

E15.5 The instar shown in Figure E15.2 is to be used to recognize a vector.

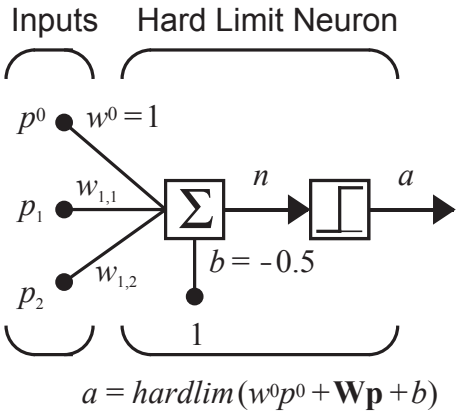
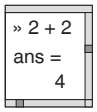


Figure E15.2 Vector Recognizer



i. Train the network with the instar rule on the following training sequence. Apply the instar rule to the second input's weights only (which should be initialized to zeros), using a learning rate of 0.6. The other weight and the bias are to remain constant at the values in the figure. (You may wish to use MATLAB to perform the calculations.)

$$\left\{p^0(1) = 1, \mathbf{p}(1) = \begin{bmatrix} 0.174 \\ 0.985 \end{bmatrix}\right\} \left\{p^0(2) = 0, \mathbf{p}(2) = \begin{bmatrix} -0.174 \\ 0.985 \end{bmatrix}\right\}$$

$$\left\{p^0(3) = 1, \mathbf{p}(3) = \begin{bmatrix} 0.174 \\ 0.985 \end{bmatrix}\right\} \left\{p^0(4) = 0, \mathbf{p}(4) = \begin{bmatrix} -0.174 \\ 0.985 \end{bmatrix}\right\}$$

$$\left\{p^0(5) = 1, \mathbf{p}(5) = \begin{bmatrix} 0.174 \\ 0.985 \end{bmatrix}\right\} \left\{p^0(6) = 0, \mathbf{p}(6) = \begin{bmatrix} -0.174 \\ 0.985 \end{bmatrix}\right\}$$

ii. What were your final values for $\mathbf{W}$?

iii. How do these final values compare with the vectors in the training sequence?

iv. What magnitude would you expect the weights to have after training, if the network were trained for many more iterations of the same training sequence?

E15.6 Consider the instar network shown in Figure E15.3. The training sequence for this network will consist of the following inputs:

$$\left\{p^0(1) = 0, \mathbf{p}(1) = \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right\}, \left\{p^0(2) = 1, \mathbf{p}(2) = \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right\}, \ldots .$$

These two sets of inputs are repeatedly presented to the network until the weight matrix $\mathbf{W}$ converges.

i. Perform the first eight iterations of the instar rule, with learning rate $\alpha = 0.25$. Assume that the initial $\mathbf{W}$ matrix is set to

$$\mathbf{W} = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

ii. Display the results of each iteration of the instar rule in graphical form (as in Figure 15.5).

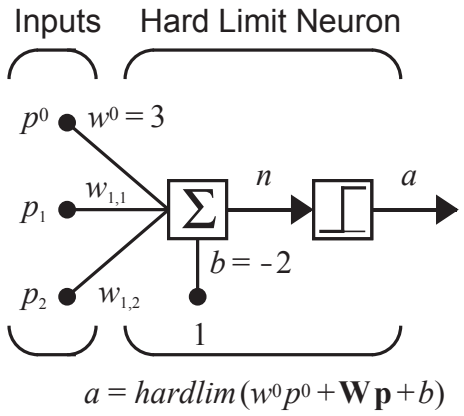


Figure E15.3 Instar Network for Exercise E15.6

E15.7 Draw a diagram of a network capable of recognizing three different four-element vectors (of ±1 values) when given different stimuli (of value 1).

i. How many inputs does your network have? How many outputs? What transfer function did you use?

ii. Choose values for the network's weights so that it can recognize each of the following vectors:

$$\mathbf{p}_1 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \qquad \mathbf{p}_2 = \begin{bmatrix} -1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \qquad \mathbf{p}_3 = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

iii. Choose an appropriate value for the biases. Explain your choice.

iv. Test the network with one of the vectors above. Was its response correct?

v. Test the network with the following vector.

$$\mathbf{p}_1 = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

Why did it respond the way it did?

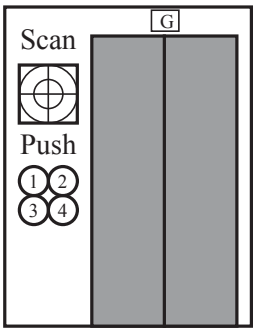
E15.8 This chapter included an example of a recognition network that initially used a visual system to identify oranges. At first the network needed the visual system to tell it when an orange was present, but eventually it learned to recognize oranges from sensor measurements.

i. Let us replace the visual system with a person. Initially, the network would depend on a person to tell it when an orange was present. Would you consider the network to be learning in a supervised or unsupervised manner?

ii. In what ways would the input from a person resemble the targets used to train supervised networks in earlier chapters?

iii. In what ways would it differ?

E15.9 The network shown in Figure E15.4 is installed in an elevator used by three senior executives in a plush high-security corporate building. It has buttons marked '1' through '4' for four floors above the ground floor. When an executive enters the elevator on the ground floor, it determines which person it is with a retinal scan, and then uses the network to select the floor where that person is most likely to go to. If the guess is incorrect, the person can push a different button at any time, but if the network is correct, it will save an important executive the effort of pushing a button.



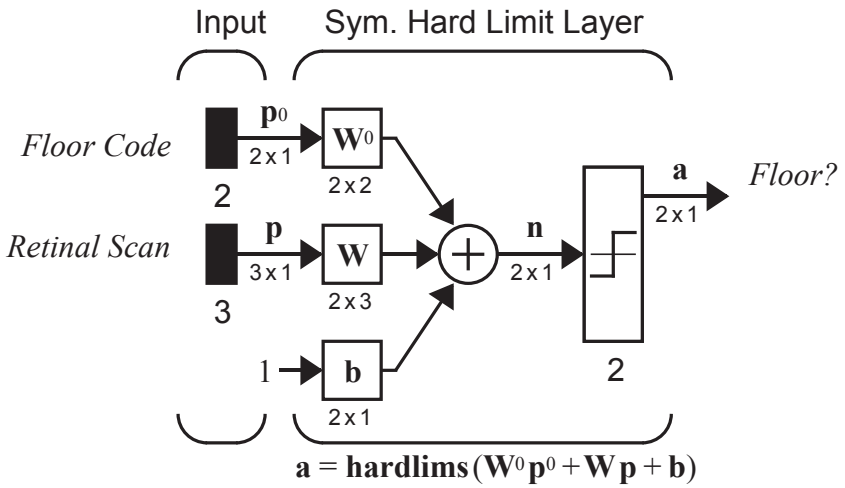
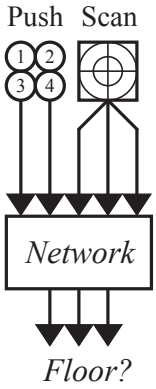


Figure E15.4 Elevator Network



The network's input/output function is

$$\mathbf{a} = \mathbf{hardlims}(\mathbf{W}^0\mathbf{p}^0 + \mathbf{W}\mathbf{p} + \mathbf{b}).$$

The first input $\mathbf{p}^0$ provides the network with a floor code, if a button has been pushed.

$$\mathbf{p}_1^0 = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \text{ (1st floor)} \qquad \mathbf{p}_2^0 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \text{ (2nd floor)}$$

$$\mathbf{p}_3^0 = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \text{ (3rd floor)} \qquad \mathbf{p}_4^0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ (4th floor)}$$

If no button is pushed, then no code is given.

$$\mathbf{p}_0^0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ (no button pushed)}$$

The first input is weighted with an identity matrix, and the biases are set to -0.5, so that if a button is pushed the network will respond with its code.

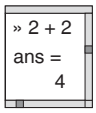
$$\mathbf{W}^0 = \mathbf{I}, \ \mathbf{b} = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix}$$

The second input is always available. It consists of three elements that represent the three executives:

$$\mathbf{p}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ (President)}, \quad \mathbf{p}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ (Vice-President)}, \quad \mathbf{p}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \text{ (Chairman)}.$$

The network learns to recall the executives' favorite floors by updating the second set of weights using the outstar rule (using a learning rate of 0.6). Initially those weights are set to zero:

$$\mathbf{W} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$



i. Use MATLAB to simulate the network for the following sequence of events:

President pushes '4', Vice-President pushes '3',
Chairman pushes '1', Vice-President pushes '3',
Chairman pushes '2', President pushes '4'.

In other words, train the network on the following sequence:

$$\{\mathbf{p}^0 = \mathbf{p}_4^0, \mathbf{p} = \mathbf{p}_1\}, \{\mathbf{p}^0 = \mathbf{p}_3^0, \mathbf{p} = \mathbf{p}_2\}, \{\mathbf{p}^0 = \mathbf{p}_1^0, \mathbf{p} = \mathbf{p}_3\},$$

$$\{\mathbf{p}^0 = \mathbf{p}_3^0, \mathbf{p} = \mathbf{p}_2\}, \{\mathbf{p}^0 = \mathbf{p}_2^0, \mathbf{p} = \mathbf{p}_3\}, \{\mathbf{p}^0 = \mathbf{p}_4^0, \mathbf{p} = \mathbf{p}_1\}.$$

ii. What are the final weights?

iii. Now continue simulating the network on these events:

President does not push a button,
Vice-President does not push a button,
Chairman does not push a button.

iv. Which floors did the network take each executive to?

v. If the executives were to push the following buttons many times, what would you expect the resulting weight matrix to look like?

President pushes '3',
Vice-President pushes '2',
Chairman pushes '4'.