Chapter 8

Plasticity and Learning

8.1 Introduction

Activity-dependent synaptic plasticity is widely believed to be the basic phenomenon underlying learning and memory. Activity-dependent modification of synapses is also thought to play a crucial role in the development of neural circuits. To understand the functional and behavioral significance of activity-dependent synaptic plasticity, we must study how experience and training modify synapses, and how these synaptic modifications change patterns of neuronal firing to affect behavior. Experimental work has shown how neuronal activity can affect synaptic strength through processes such as long-term potentiation (LTP) and long-term depression (LTD). Theoretical work on synaptic plasticity derives from learning rules, based on these experimental results, that propose how training and experience change synaptic weights. Synaptic learning rules have been applied to a wide range of tasks including auto- and hetero-associative memory, pattern recognition, storage and recall of temporal sequences, and function approximation.

In 1949, Donald Hebb conjectured that if input from neuron A often contributes to the firing of neuron B, the synapse from A to B should be strengthened. Hebb's rule, sometimes paraphrased as 'neurons that fire together, wire together', forms the basis of much of the research done on the role of synaptic plasticity in learning and memory. Hebb suggested that activity-dependent synaptic modification could produce neuronal assemblies that reflect the relationships experienced during learning. For example, neurons that fire together during training due to an association between a stimulus and a response, would develop strong interconnections. Subsequently, the activation of a subset of these neurons due to presentation of the stimulus would generate strong enough synaptic drive to activate the remaining neurons representing the response.

Experimental work has revealed processes of long-term synaptic strengthening (LTP) that capture the spirit of Hebb's rule, and long-term synaptic weakening (LTD) as well. A variety of different types of LTP and LTD have been seen in slices from a number of brain regions including hippocampus, neocortex, and cerebellum. Figure 8.1 shows an example from intracellular recordings of a pair of cortical pyramidal cells in a slice experiment. At the time marked by the arrow (time zero) in the figure, a pre- and postsynaptic pair of cells was stimulated to fire in four different temporal orderings. For the traces marked by open symbols, the presynaptic neuron fired an action potential and evoked an epsp in the postsynaptic neuron either 10 or 100 ms before the postsynaptic neuron fired an action potential. The traces marked by solid symbols denote the reverse ordering in which the presynaptic neuron fired either 10 or 100 ms after the postsynaptic neuron. Separations of 100 ms had no long lasting effect on the synaptic strength as measured by the amplitude of the evoked epsp in test trials more than 40 minutes after the paired stimulation. In contrast, the 10 ms delays produced effects that built up to a maximum over a 10 to 20 minute period and then lasted for the duration of the experiment. Pairing a presynaptic action potential with a postsynaptic action potential 10 ms later produced LTP. LTD occurred when the presynaptic action potential followed the postsynaptic action potential by 10 ms. Thus, as Hebb conjectured, the synapse is strengthened when a presynaptic action potential precedes a postsynaptic action potential and can thereby be interpreted as contributing to it. In addition, when the order is reversed and the presynaptic action potential could not have contributed to the postsynaptic response, the synapse is weakened.

Most studies of network learning involve simplified descriptions of synaptic plasticity and network dynamics. This work overlaps with the field of artificial neural networks, which is built on the general idea that collections of simple and fairly undifferentiated processing elements can organize their interconnections through learning to meet the demands of particular tasks. Studies of artificial neural networks have revealed many of the computational functions that can be achieved through synaptic modification.

Models in which the activity of each neuron is described by a firing rate, as in chapter 7, are particularly useful for studying the role of synaptic plasticity. Synaptic modification is included in these models by specifying how a parameter w that characterizes the strength of a synapse between two neurons changes as a function of the pre- and postsynaptic levels of activity, for example the pre- and postsynaptic firing rates. To avoid the use of the subscripts pre and post throughout this chapter, we use the letter u to denote the presynaptic level of activity and v for the postsynaptic level of activity. Sometimes u and v stand for fairly abstract measures of activity, and they are not required to be greater than zero. At other times, u and v can be considered as pre- and postsynaptic firing rates.

In general, rules for synaptic modification take the form of a differential

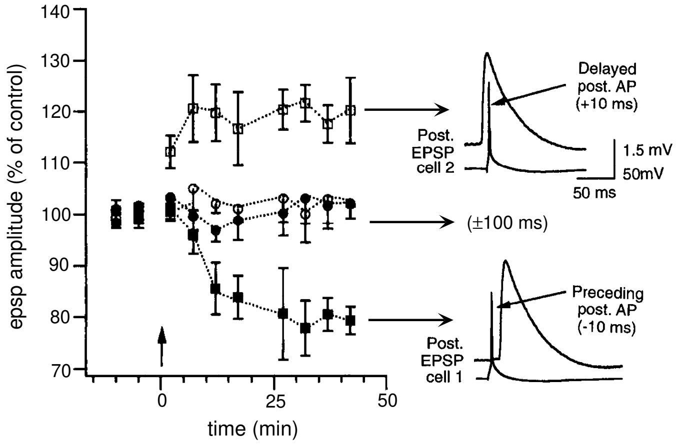

Figure 8.1: LTP and LTD produced by paired stimulation with different timing. The traces show the amplitude of epsp's evoked by stimulation of the presynaptic neuron plotted as a percentage of the amplitude prior to paired stimulation. At the time indicated by the arrow, 50 to 75 paired stimulations of the presynaptic and postsynaptic neurons were performed with the following temporal ordering: open squares - presynaptic spike preceded postsynaptic spike by 10 ms; open circles - presynaptic spike preceded postsynaptic spike by 100 ms; filled squares - presynaptic spike followed postsynaptic spike by 10 ms; filled circles - presynaptic spike followed postsynaptic spike by 100 ms. (Adapted from Markram *et al*, 1997)

equation describing the rate of change of the synaptic weight as a function of the pre- and postsynaptic firing rates,

$$\tau \frac{dw}{dt} = F(u, v)\,, \tag{8.1}$$

or a discrete update rule,

$$w \to w + \epsilon F(u, v)\,. \tag{8.2}$$

The function F determines the nature of the learning rule, and τ and ϵ are parameters that control the rate of synaptic change. The simplest learning rule inspired by Hebb's conjecture has $F = uv$, so that simultaneous pre- and postsynaptic firing increases synaptic strength. There are many variants of this basic rule, several of which are discussed in this chapter.

Studies of plasticity and learning involve analyzing how synapses are affected by activity over the course of a training period during which a given synaptic learning rule is applied. The outcome of any training procedure depends not only on the synaptic learning rule, but also on the input and output activity during the training period. Training procedures are classified depending on whether network output during training is determined by the input activity according to the normal network dynamics, or is specified or clamped by some external agent. The first case, called unsupervised (or sometimes self-supervised) learning, corresponds to a self-organizing network that develops a particular input-output map solely on the basis of the network dynamics, synaptic learning rule, and statistics of the input data presented during training. The second case, supervised learning, involves a 'teacher' forcing a desired pattern of output activity upon the network during training. In either case, the learning rule is only applied during the training period. After training, the network is governed solely by its intrinsic dynamics. A form of learning intermediate between unsupervised and supervised, reinforcement learning, is considered in chapter 10.

8.2 Unsupervised Hebbian Learning

An important computational interpretation of Hebbian learning can be derived by studying a single unit with output v driven by a set of inputs u_a through synapses of strength w_a, with $a = 1, 2, \ldots, N$. The output v is determined by the linear rule

$$v = \sum_{a=1}^{N} w_a u_a = \mathbf{w} \cdot \mathbf{u} \, . \qquad (8.3)$$

The second equality introduces a simple geometric interpretation of the input-output relationship. If we think of the set of inputs as a vector $\mathbf{u}$ (with components u_a) in an N-dimensional space, and likewise represent the weights by a vector $\mathbf{w}$, the sum in the first equality of equation 8.3 can be written as the dot product $\mathbf{w} \cdot \mathbf{u}$. This means that v can be interpreted as the projection of the input vector $\mathbf{u}$ onto the weight vector $\mathbf{w}$. The weights thus define a projection axis for inputs to this linear unit.

An unsupervised training session for this network is a period of time during which different input vectors are repeatedly presented while a Hebbian learning rule is applied to the weights $\mathbf{w}$. Each time an input vector $\mathbf{u}$ is presented, the weights are modified according to the rule 8.2 with $F = vu$ so that

$$w_a \rightarrow w_a + \epsilon v u_a = w_a + \epsilon \left(\sum_{b=1}^{N} w_b u_b \right) u_a \, . \qquad (8.4)$$

The parameter ϵ determines the learning rate. Note that we have used equation 8.3 for v in the second equality because the training is unsupervised. This learning rule can also be written in vector notation as

$$\mathbf{w} \rightarrow \mathbf{w} + \epsilon v \mathbf{u} = \mathbf{w} + \epsilon (\mathbf{w} \cdot \mathbf{u}) \mathbf{u} \qquad (8.5)$$

The learning rule 8.4 depends on the input used during training through the product $u_a u_b$. The total weight change produced by all of the input vectors presented during training is determined by the accumulation of the changes given by equation 8.4. If ϵ is small enough, the resulting change can be approximated by replacing the product $u_a u_b$ in 8.4 by its average value over all of the different input patterns, which is the correlation matrix of the input vector $\mathbf{u}$ denoted by

$$Q_{ab} = \langle u_a u_b \rangle \quad \text{or} \quad \mathbf{Q} = \langle \mathbf{uu} \rangle \,. \tag{8.6}$$

In the right hand equation $\mathbf{Q}$ and $\mathbf{uu}$ represent N by N matrices with elements Q_{ab} and $u_a u_b$ respectively. By averaging over input vectors, the rule 8.4 can be written in terms of the correlation matrix as

$$w_a \rightarrow w_a + \epsilon \sum_{b=1}^{N} Q_{ab} w_b \tag{8.7}$$

or, in vector form,

$$\mathbf{w} \rightarrow \mathbf{w} + \epsilon \mathbf{Q} \cdot \mathbf{w} = (\mathbf{I} + \epsilon \mathbf{Q}) \cdot \mathbf{w} \,. \tag{8.8}$$

Here $\mathbf{I}$ is the identity matrix, and $\mathbf{Q} \cdot \mathbf{w}$ denotes the vector with components $\sum_b Q_{ab} w_b$. Because of the presence of the input correlation matrix $\mathbf{Q}$, equation 8.8 is called a correlational learning rule.

If the weight vector at the beginning of the training period is $\mathbf{w}(0)$, and if the learning rule 8.8 is applied P times, the result will be

$$\mathbf{w} = (\mathbf{I} + \epsilon \mathbf{Q})^P \cdot \mathbf{w}(0) \,. \tag{8.9}$$

This result can be analyzed using the method of matrix diagonalization which involves expressing $\mathbf{w}$ in terms of a special set of basis vectors called eigenvectors. The eigenvectors, denoted by $\mathbf{e}_\mu$ with $\mu = 1, 2, \cdots, N$, are an orthogonal set of unit length vectors, $\mathbf{e}_\mu \mathbf{e}_\nu = \delta_{\mu\nu}$, which satisfy

$$\mathbf{Q} \cdot \mathbf{e}_\mu = \lambda_\mu \mathbf{e}_\mu \,. \tag{8.10}$$

The numbers λ_μ are the eigenvalues for the eigenvectors $\mathbf{e}_\mu$ and, for a correlation matrix, they are all real and non-negative. In the cases we consider, the eigenvalues will all be positive.

Any N-dimensional vector can be represented using the N eigenvectors as a basis. For example, we can write

$$\mathbf{w}(0) = \sum_{\mu=1}^{N} \left(\mathbf{w}(0) \cdot \mathbf{e}_\mu \right) \mathbf{e}_\mu \tag{8.11}$$

Writing $\mathbf{w}(0)$ as a sum of eigenvectors turns matrix multiplication into ordinary scalar multiplication, making matrix calculations easier. Substituting the expansion 8.11 into 8.9 and using 8.10, we find that, after P iterations,

$$\mathbf{w} = \sum_{\mu=1}^{N} \left(1 + \epsilon \lambda_\mu\right)^P \left(\mathbf{w}(0) \cdot \mathbf{e}_\mu \right) \mathbf{e}_\mu \,. \tag{8.12}$$

Because the eigenvalues λ_μ are positive for all μ values, the factors $(1+\epsilon\lambda_\mu)^P$ in 8.12 all grow exponentially as functions of P. Furthermore, for large P, the term with the largest value of λ_μ (assuming it is unique) becomes much larger than all the other terms and dominates the sum for $\mathbf{w}$. We assign the largest eigenvalue and its corresponding eigenvector, sometimes called the principal eigenvector, the label $\mu = 1$. Thus, for large P, $\mathbf{w} \propto \mathbf{e}_1$ to a good approximation. After training, the response of the unit to an arbitrary input vector $\mathbf{u}$ will be well-approximated by

$$v \propto \mathbf{e}_1 \cdot \mathbf{u} \,. \tag{8.13}$$

Thus, Hebbian learning produces a set of weights that project the input vector onto the principal eigenvector of the correlation matrix of the inputs used during training.

The proportionality sign in equation 8.13 hides the large factor $(1+\epsilon\lambda_1)^P$. We next discuss methods for normalizing the synaptic weights to prevent the uncontrolled growth that this factor indicates.

Constraints on the Growth of Synaptic Weights

The magnitude of $\mathbf{w}$ increases without bound for correlation-based Hebbian learning because the eigenvalues of $\mathbf{Q}$ are positive. Unfettered growth is a result of the positive feedback inherent in Hebbian learning. Two mechanisms used to limit growth, saturation and normalization, are analyzed in this section. Another mechanism, the sliding threshold, is discussed in a later section on the BCM rule. Experimental evidence for various mechanisms that may underlie constraints on synaptic strength is accumulating (Turrigiano, 1998).

Saturation

Synaptic saturation imposes strict bounds on the values of the weights. One sensible constraint to impose is $w_a \geq 0$ for all a, since LTP and LTD cannot change an excitatory synapse into an inhibitory synapse. An upper bound $w_a \leq w_{\text{max}}$ for all a is also supported by LTP experiments. The simplest implementation of saturation is to set any weight that would cross a saturation bound due to application of a learning rule to the limiting value. In such a scheme, the weights grow exponentially only until one or more weights saturate and the constraints start affecting growth. A weight that goes to zero may also be interpreted as a synapse that has been anatomically removed so that it cannot later recover to a nonzero value.

Figures 8.2A and B show examples of the Hebbian development of two weights constrained to lie between zero and one. For the correlation matrix used in this example, the fastest growing mode was $\mathbf{e}_1 = (1, -1)$. An analysis that ignores saturation would thus predict that one weight

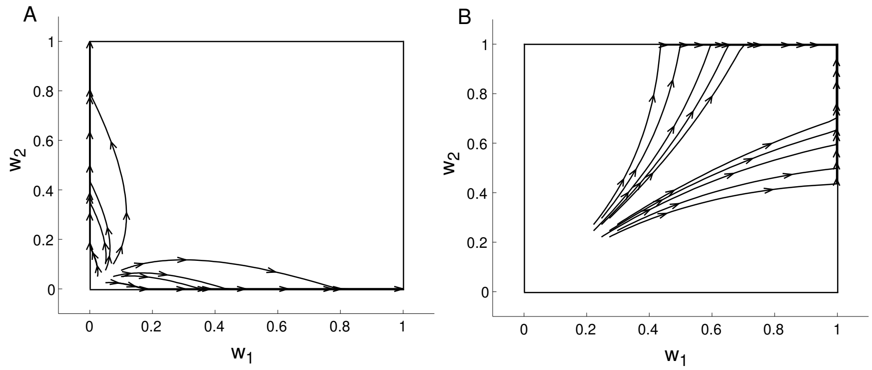

Figure 8.2: Hebbian weight dynamics with saturation. The correlation matrix of the input vectors had diagonal elements equal to 1 and off-diagonal elements of -0.4. The weights saturate at 0 and 1. A) The eigenmode $\mathbf{e}_1 = (1, -1)$ dominates the dynamics if the initial values of the weights are small. This makes the weight vector move to the corners $(1, 0)$ or $(0, 1)$. B) Starting the weights with more positive values allows saturation to occur at the corner $(1, 1)$. (Adapted from MacKay & Miller, 1990)

should grow in the positive direction and the other in the negative direction. Given the constraints, this would suggest that $(1, 0)$ and $(0, 1)$ are the most likely final configurations. This analysis gives the correct answer for figure 8.2A, but not for figure 8.2B. These figures differ only in the initial values of the synaptic weights. In figure 8.2B the final state is $\mathbf{w} = (1, 1)$ because the weights hit the saturation boundary before the exponential growth was large enough to allow the principal eigenvector to dominate. A useful approach in the analysis of Hebbian plasticity is to determine the pattern of synaptic weights with the fastest growth rate, as we did in the previous section, and assume that the constrained weights will be proportional to this fastest growing mode. Figure 8.2B shows that saturation can sometimes invalidate such an analysis.

Hebbian learning with an upper saturation constraint has the undesirable feature that all (or all but one) of the weights tend to reach the upper limit and thus to become equal. This is often inadequate for a specific computational goal and more competition must be introduced between the weights by using the subtractive normalization constraint discussed below.

Normalization

Unlike saturation, in which each weight is individually constrained, normalization constrains a combination of weights. One such constraint is to hold the length of the weight vector fixed by setting $\sqrt{\mathbf{w} \cdot \mathbf{w}} = |\mathbf{w}| = 1$. Another constraint is to maintain a constant value for the sum of all the weights, although this only limits growth if the weights are also restricted to be non-negative by a saturation constraint. Normalization introduces competition into the synaptic modification process by forcing some weights to decrease when others grow. Weights can be forced to satisfy a normalization constraint through multiplication or subtraction. The effects of the two forms of normalization can be quite different. Multiplicative normalization is used to hold the length of the weight vector fixed, and subtractive normalization is typically used when the sum of weights is constrained.

Multiplicative normalization maintains $|\mathbf{w}| = 1$ by replacing equation 8.5 with

$$\mathbf{w} \rightarrow \frac{\mathbf{w} + \epsilon v \mathbf{u}}{|\mathbf{w} + \epsilon v \mathbf{u}|} . \tag{8.14}$$

The denominator assures that $|\mathbf{w}| = 1$. To first order in ϵ, equation 8.14 is equivalent to

$$\mathbf{w} \rightarrow \mathbf{w} + \epsilon (v\mathbf{u} - v^2 \mathbf{w}) . \tag{8.15}$$

This variant of the standard Hebbian learning rule is called the Oja rule (1982). Note that it only involves information that is local to the synapse being modified, namely the pre- and postsynaptic activities and the local synaptic weight. Averaging equation 8.15 over the input vectors presented during an unsupervised training period leads to the averaged rule

$$\mathbf{w} \rightarrow \mathbf{w} + \epsilon (\mathbf{Q} \cdot \mathbf{w} - (\mathbf{w} \cdot \mathbf{Q} \cdot \mathbf{w}) \mathbf{w}) \tag{8.16}$$

where $\mathbf{w} \cdot \mathbf{Q} \cdot \mathbf{w} = \sum_{a,b} w_a Q_{ab} w_b$. The change in $\mathbf{w}$ imposed by this equation goes to zero when $\mathbf{w}$ is an eigenvectors of $\mathbf{Q}$ normalized to unit length (see exercise 2). Because the principal eigenvector grows fastest, the Oja rule typically sets the weight vector equal to $\mathbf{e}_1$, correctly normalized to unit length. This fixes the problem of the proportionality constant that was skirted in our earlier discussion.

When the weights are constrained to be non-negative, an alternative to multiplicative normalization is to maintain the sum of the weights at a constant value. It is convenient to express this sum as $\sum w_a = \mathbf{n} \cdot \mathbf{w}$ where $\mathbf{n}$ is an N-dimensional vector with all its components equal to one. Holding $\mathbf{n} \cdot \mathbf{w}$ constant can be achieved through a subtractive process by replacing equation 8.5 with

$$\mathbf{w} \rightarrow \mathbf{w} + \epsilon \left(v\mathbf{u} - \frac{v(\mathbf{n} \cdot \mathbf{u})\mathbf{n}}{N} \right) . \tag{8.17}$$

To see that this rule does not change $\mathbf{n} \cdot \mathbf{w}$, take the dot product of the second term with $\mathbf{n}$ and note that $v\mathbf{n} \cdot \mathbf{u} - v(\mathbf{n} \cdot \mathbf{u})\mathbf{n} \cdot \mathbf{n}/N = 0$ because $\mathbf{n} \cdot \mathbf{n} = N$. Unlike the Oja rule, this rule is not local. The change in the weight of each synapse depends on the sum of the presynaptic activities at all other synapses. Averaging equation 8.17 over the distribution of input patterns, the averaged learning rule with subtractive normalization is

$$\mathbf{w} \rightarrow \mathbf{w} + \epsilon \left(\mathbf{Q} \cdot \mathbf{w} - \frac{(\mathbf{w} \cdot \mathbf{Q} \cdot \mathbf{n})\mathbf{n}}{N} \right) . \tag{8.18}$$

Subtractive normalization, combined with a saturation bound of zero, typically produces just a single nonzero weight. This can be avoided by including an additional upper saturation limit at an appropriately chosen maximum value. Thus, both upper and lower saturation bounds are often used with subtractive normalization.

The multiplicative and subtractive normalization procedures we have presented yield different results. Multiplicative normalization is equalizing in that it reduces large weights by more than it reduces small weights. Subtractive normalization reduces all weights by the same numerical amount and thus induces more competition. The competitive nature of subtractive normalization is important for modeling the activity-dependent development of receptive field structures in primary visual cortex.

It is easier to analyze a learning rule if the constraints that limit growth are ignored. Sometimes the effect of the constraint is relatively benign, for example it may limit growth of the length of the weight vector while having little effect on its direction. In this case, the direction of weight growth established by the principal eigenvector of the correlation matrix without the constraints is a good indicator of the direction along which the final weight vector will lie. This is more likely to be true of a multiplicative than a subtractive normalization scheme. Multiplicative normalization does not change the ratios of the growth rates of the different eigenmodes because all the weights are multiplied by the same factor. Subtractive normalization can sometimes prevent the weights from growing in the direction of the principal eigenvector. They then end up being proportional to the eigenvector with the largest eigenvalue among the group of eigenvectors allowed to grow by the constraint.

Development of Simple Cell Receptive Fields

Hebbian learning rules have been quite successful in modeling the activity-dependent development of response properties of cells in primary visual cortex. We focus here on two fundamental properties of simple cells, ocular dominance and orientation selectivity. A more extended discussion of these ideas appears in chapter 9.

We saw in chapter 2 that simple cells of the primary visual cortex respond selectively to the spatial frequency, spatial phase, and orientation of an image within their receptive fields. Neurons in primary visual cortex receive input from both eyes, but they tend to be dominated by input from one eye or the other, especially in layer 4 which receives extensive innervation from the LGN. Areas with left- or right-eye ocular dominance alternate across the cortex in bands. At an early stage of development, neurons in layer 4 are equally innervated by LGN inputs from both eyes. Experiments have shown that the subsequent development of ocular dominance is affected by activity.

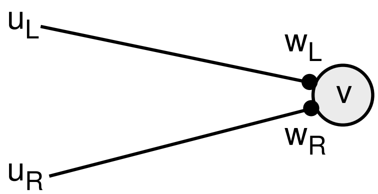

Figure 8.3: A two-input model of ocular dominance. LGN inputs from the left and right eyes activate a single output unit through weights w_L and w_R respectively.

Ocular Dominance

We first consider a highly simplified version of the model of Miller, Keller, and Stryker (1989) for ocular dominance development. The simplicity of our version of the model helps to reveal its basic features. In the simplified model, a single simple cell receives input from just two LGN afferents (figure 8.3); one, u_L, from the left eye, and the other, u_R, from the right eye. Both afferents cover the same region of the visual field. Two synaptic weights $\mathbf{w} = (w_1, w_2)$ control the strength of these projections, and the output of the layer 4 simple cell, v, is determined by

$$v = w_L u_L + w_R u_R \,. \tag{8.19}$$

The weights are constrained by saturation to nonnegative values. Initially, the weights for the left- and right-eye inputs are set to approximately equal values to match the roughly equal innervation from each eye seen early in development. As we will see, Hebbian learning combined with a subtractive normalization constraining the sum of the weights, leads to the weight vector $\mathbf{w} = (1, 0)$ or $(0, 1)$ after training, reflecting dominance by one eye or the other.

We begin by studying what happens if an unconstrained Hebbian rule is applied during the development period. The result depends on the correlation matrix between the left- and right-eye inputs,

$$\mathbf{Q} = \langle \mathbf{uu} \rangle = \begin{pmatrix} \langle u_L u_L \rangle & \langle u_L u_R \rangle \\ \langle u_R u_L \rangle & \langle u_R u_R \rangle \end{pmatrix} = \begin{pmatrix} q_S & q_D \\ q_D & q_S \end{pmatrix} . \tag{8.20}$$

We use the subscripts S and D to denote same- and different-eye correlations, and we assume that the same eye correlations are identical for the two eyes. The eigenvectors of this correlation matrix are $\mathbf{e}_1 = (1, 1)$ and $\mathbf{e}_2 = (1, -1)$ with eigenvalues $\lambda_1 = q_S + q_D$ and $\lambda_2 = q_S - q_D$ respectively. Positive correlations are likely to exist between the two eyes after eye opening has occurred. If $q_D > 0$, $\mathbf{e}_1$ has the largest eigenvalue and unconstrained Hebbian learning would produce a weight vector $\mathbf{w} \propto \mathbf{e}_1 = (1, 1)$. This predicts equal innervation from both eyes and is not the result we seek. Of course, it also predicts unbounded growth of the weights, another undesirable result. Even if $q_D < 0$, the mode proportional to $\mathbf{e}_1$ may tend to dominate because the weights start out approximately equal, giving this mode a large initial value.

Both of these problems can be fixed by including a normalization constraint on the sum of weights, $w_L + w_R = 1$ and using the Hebbian rule with subtractive normalization, equation 8.18. This rule completely eliminates the growth of the weight vector in the direction of $\mathbf{e}_1$. On the other hand, it has no effect on growth in the direction $\mathbf{e}_2$ because $\mathbf{e}_2 \cdot \mathbf{n} = 0$, so this growth has no effect on the sum of weights. Thus, with subtractive normalization, the weight vector grows parallel (or anti-parallel) to the direction $\mathbf{e}_2 = (1, -1)$. The direction of this growth depends on initial conditions through the value of $\mathbf{w}(0) \cdot \mathbf{e}_2 = w_L(0) - w_R(0)$. If this is positive ($w_L > w_R$), w_L will increase and w_R will decrease; if it is negative ($w_L < w_R$), w_R will increase and w_L will decrease. Eventually, one of the weights will hit the saturation limit of zero, and the final values for the weights will be either $(1, 0)$ or $(0, 1)$ reflecting left- or right-eye dominance.

This simple model shows that ocular dominance can arise from Hebbian learning if there is sufficient competition between the growth of the left- and right-eye weights. Competition was implemented through subtractive normalization. Multiplicative normalization would not be effective for this example because it affects growth in the directions $\mathbf{e}_1$ and $\mathbf{e}_2$ equally. Thus, it cannot prevent growth in the $\mathbf{e}_1$ direction that produces binocular innervation from dominating over growth in the $\mathbf{e}_2$ direction that produces ocular dominance.

Orientation Selectivity

The model of Hubel and Wiesel (1962) for generating an orientation selective simple cell response by summing linear arrays of alternating ON and OFF LGN inputs was presented in chapter 2 (see figure 2.24A). Miller (1994) has argued that such an input pattern can arise from Hebbian learning on the basis of correlations and anticorrelations between the responses of different LGN cells. As in the previous example, we consider a single simple cell receiving input from ON-center and OFF-center cells of the LGN, but, in this case, we assume that all input is from a single eye. The analysis of the last section showed that subtractive normalization can eliminate equal growth of two weights. On the basis of this result, we let the variable u_a represent the difference between the firing rates of ON- and OFF-center LGN cells with receptive fields centered at the same spatial location. In this case, we do not restrict the sign of the synaptic weights. We interpret a positive synaptic weight w_a as representing direct excitation from an ON-center LGN cell and indirect feedforward inhibition from an OFF-center cell, both at location a. A negative weight represents the opposite, inhibition from the ON-center cell and excitation from the OFF-center cell. This interpretation simplifies the analysis, and it gives results similar to a more complete description in which ON- and OFF-center cells are not combined and an appropriate subtractive normalization is applied.

In this model of receptive field development, constraints play a less important role than in the ocular dominance model discussed above. Following Miller (1990 & 1996) and Mackay and Miller (1990) we will ignore constraints on weight growth in our analysis. After presenting the simplified analysis we will show that the results agree with those obtained through simulations in which constraints are included.

To analyze Hebbian learning in this case, we replace the discrete input label a with a continuous two dimensional vector $\vec{a}$ that represents the location of the center of the receptive field of a given ON-OFF cell pair. The input to the model simple cell from such a pair is denoted by $u(\vec{a})$. Similarly, we replace the weight w_a by the function $w(\vec{a})$. Finally, we replace sums over input cells by integrals over the locations of their receptive field centers, making the replacement $\sum_a \to \rho \int d\vec{a}$. The factor ρ is the number of LGN cell pairs per unit area of retinal surface defined so that $\sum_a = \rho \int d\vec{a} = N$, the number of LGN pairs in the model.

Using this continuous notation, equation 8.3 for the output of the simple cell is replaced by

$$v = \rho \int d\vec{a}\, w(\vec{a}) u(\vec{a}) \,. \tag{8.21}$$

In this continuous notation, the input correlation matrix becomes a function $Q(\vec{a} - \vec{b}) = \langle u(\vec{a}) u(\vec{b}) \rangle$, and the input-averaged Hebbian learning rule 8.7 is now

$$w(\vec{a}) \to w(\vec{a}) + \epsilon\rho \int d\vec{b}\, Q(\vec{a} - \vec{b}) w(\vec{b}) \,. \tag{8.22}$$

The fastest growing eigenmode, which will determine the final value of the weight function if constraints do not interfere, now satisfies the equation

$$\int d\vec{b}\, Q(\vec{a} - \vec{b}) e(\vec{b}) = \lambda e(\vec{a}) \tag{8.23}$$

with the largest possible value of λ.

The solutions of equation 8.23, assuming that Q depends only on the magnitude of the vector difference, $|\vec{a} - \vec{b}|$, are functions of the form

$$e(\vec{a}) = \cos(\vec{k} \cdot \vec{a} - \phi) \tag{8.24}$$

with arbitrary values of $\vec{k}$ and ϕ. The corresponding eigenvalues are $\lambda = \tilde{Q}(\vec{k})$, where $\tilde{Q}(\vec{k})$ is the Fourier transform of Q. The principal eigenvector is thus a function of the form 8.24 with $\vec{k}$ chosen to maximize the Fourier transform $\tilde{Q}(\vec{k})$.

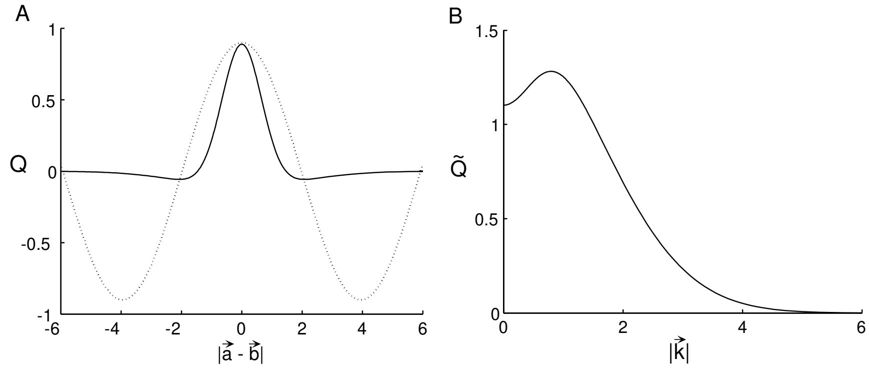

Figure 8.4: Hypothesized ON/OFF-center cell correlation function. A) The solid line is the correlation function Q given by 8.25 with $A = 1$, $B = 1/9$, $\sigma_A = 0.435$, and $\sigma_B = 3.92$. The dotted line is the principal eigenvector for this correlation function plotted on the same scale. Distance units are arbitrary. B) The Fourier transform $\tilde{Q}$ of Q. This is also given by the difference of two Gaussians. Units of $|\vec{k}|$ are arbitrary. The spatial wavelength of the cosine function in A is determined by the $|\vec{k}|$ value at which this Fourier transform peaks.

Miller (1994) has argued, on the basis of the center-surround structure of LGN receptive fields, that the correlation function $Q(\vec{a} - \vec{b})$ can be approximated by a difference of two Gaussian functions,

$$Q(\vec{a} - \vec{b}) = A \exp\left(-\frac{|\vec{a} - \vec{b}|^2}{2\sigma_A^2}\right) - B \exp\left(-\frac{|\vec{a} - \vec{b}|^2}{2\sigma_B^2}\right), \tag{8.25}$$

with $\sigma_A < \sigma_B$. This function and its Fourier transform are plotted in 8.4A and B. Because the correlation function $Q(\vec{a} - \vec{b})$ depends only on the distance between $\vec{a}$ and $\vec{b}$, the Fourier transform $\tilde{Q}(\vec{k})$ depends only on $|\vec{k}|$. The length of the vector $\vec{k}$ to be used in equation 8.24 is determined by the location of the maximum of $\tilde{Q}(\vec{k})$, but the direction of this vector and the phase angle ϕ are undetermined. All of the modes given by 8.24 with different $\vec{k}$ directions and phase angles will grow at the same rate when a Hebbian learning rule is applied to the synaptic weights.

If constraints do not interfere with the trend set up by the unconstrained Hebbian learning rule, the weight function $w(\vec{a})$ will develop to be proportional to some linear combination of the fastest growing weights. The particular linear combination that develops depends on the initial values of the weights at the beginning of the development period. If a single direction of $\vec{k}$ and value of ϕ dominates, the oscillation of the cosine function in 8.24 sets up an alternating pattern of ON-center and OFF-center cell innervation resembling that shown in figure 8.3B. This establishes orientation selective responses. Because all directions of $\vec{k}$ and ϕ values are possible, the resulting receptive fields can have any direction of orienta-

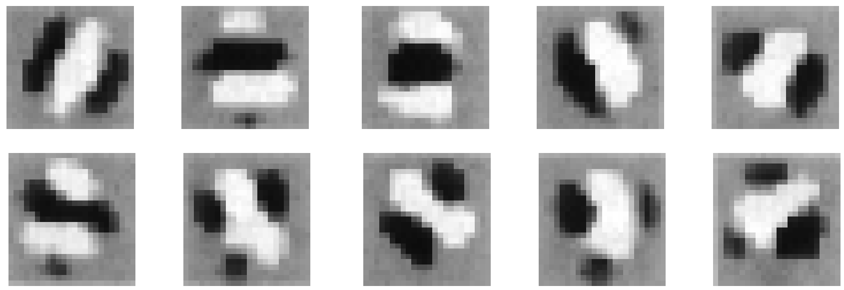

Figure 8.5: Different cortical receptive fields arising from a correlation-based developmental model. The individual plots show the weights $w(\vec{a})$ for a two-dimensional receptive field location vector $\vec{a}$ with white denoting positive values and black negative values. These correspond to regions where ON-center cells (white regions) or OFF-center cells (black regions) excite the cortical neuron. (Adapted from Miller, 1994)

tion selectivity and any value of preferred spatial phase.

In the model considered thus far, LGN cells centered at any location $\vec{a}$ can make connections with the model cortical cell. As a result, the receptive field that arises from the alternation of ON-center and OFF-center innervation has an unlimited extent. In reality, neurons in primary visual cortex only receive afferents from LGN cells centered over a finite region of the visual space. This anatomical constraint can be included in the developmental model by introducing what is called an arbor function. The arbor function, which is often taken to be Gaussian, characterizes the density of innervation from a given visual location to the simple cell being modeled. The model assumes that this density cannot be altered during the Hebbian developmental process, but that the strengths of synapses within the arbor function are modified by the Hebbian rule. This results in oriented receptive fields with limited spatial extent (figure 8.5). If the arbor function is Gaussian, and the synaptic weights are given by a single term 8.24, the result is a receptive field described by a Gabor function, as discussed in chapter 2.

Figure 8.5 shows the weights resulting from a simulation of receptive field development from a large array of ON- and OFF-center LGN afferents. These show the variety of oriented receptive field structures that can arise through a Hebbian developmental rule. More complicated receptive field structures than the ones we have derived are seen in some of the panels of figure 8.5. Theses arise because of saturation constraints that we have not considered. More details about these models are presented in chapter 9.

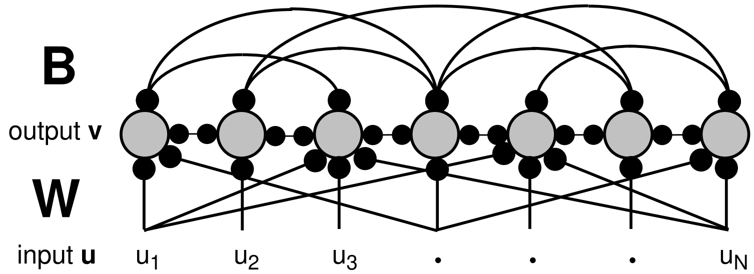

Figure 8.6: A network with multiple output units driven by feedforward synapses with weights **W**, and interconnected by recurrent synapses with weights **B**.

Multiple Output Units

Up to now, we have only considered feedforward input to a single unit. To describe multiple output units with activities v_a, for $a = 1 \ldots M$, driven by feedforward input, we replace equation 8.3 by

$$v_a = \sum_{b=1}^{N} W_{ab} u_b \quad \text{or} \quad \mathbf{v} = \mathbf{W} \cdot \mathbf{u} \,. \tag{8.26}$$

Here, as in chapter 7, W_{ab} is the strength of the synapse connecting input unit b to output unit a. The right-hand equation uses vector and matrix notation where $\mathbf{v}$ is a vector of inputs with components v_a and $\mathbf{W}$ is the M by N matrix of synaptic weights with elements W_{ab}. Note that the feedforward synaptic weights are now described by a matrix instead of a vector.

Applying the Hebb rule to an array of output neurons driven by purely feedforward input is rather trivial because each unit develops independently. A more interesting scenario is when the multiple outputs are connected to each other as in figure 8.6. The strength of the connection from output cell b to output cell a is determined by the element B_{ab} of the M by M matrix $\mathbf{B}$. A simple extension of the rule 8.26 determining the output activity is then

$$\mathbf{v} = \mathbf{W} \cdot \mathbf{u} + \mathbf{B} \cdot \mathbf{v} \,. \tag{8.27}$$

We will only consider linear models in this chapter. As discussed in chapters 9 and 11, the competition between different cortical cells introduced by nonlinear schemes can have an important impact.

It is best to think of equation 8.27 as a condition determining the stable equilibrium point of a dynamic equation such as $\tau d\mathbf{v}/dt = -\mathbf{v} + \mathbf{W} \cdot \mathbf{u} + \mathbf{B} \cdot \mathbf{v}$. As discussed in chapter 7, the stable equilibrium of this equation is determined by 8.27, provided that the real parts of the eigenvalues of $\mathbf{B}$ are all less than 1. We assume that this condition is satisfied and use the static equation 8.27, rather than this dynamic equation, because we assume that inputs are presented long enough for equilibrium to be achieved. Equation

8.27 can be solved by defining the matrix inverse $\mathbf{K} = (\mathbf{I} - \mathbf{B})^{-1}$ yielding

$$\mathbf{v} = \mathbf{K} \cdot \mathbf{W} \cdot \mathbf{u} \,. \tag{8.28}$$

In this equation, $\mathbf{K} \cdot \mathbf{W} \cdot \mathbf{u}$ is the vector with components $\sum_{b,c} K_{ab} W_{bc} u_c$.

In a network with both feedforward and recurrent connections (figure 8.6), synaptic learning rules can be applied to the feedforward weights $\mathbf{W}$, the recurrent weights $\mathbf{B}$, or both. In the following section, we study ocular dominance maps by applying a Hebbian learning rule to the feedforward weights while holding the recurrent weights fixed. If the recurrent weights are held fixed so that the output activities are determined by 8.28 when the Hebbian rule is applied to the feedforward weights, the averaged learning rule is

$$\mathbf{W} \to \mathbf{W} + \epsilon \langle \mathbf{vu} \rangle = \mathbf{W} + \epsilon \mathbf{K} \cdot \mathbf{W} \cdot \mathbf{Q} \,. \tag{8.29}$$

Here $\mathbf{vu}$ is the matrix with elements $v_a u_b$. Equation 8.29 has the same form as the single unit equation 8.8 except that both $\mathbf{K}$ and $\mathbf{Q}$ affect the growth of $\mathbf{W}$. In later sections, we consider synaptic modification of recurrent weights with both fixed and modifiable feedforward weights.

Cortical Maps

In the models of ocular dominance and orientation selectivity that we discussed, the ultimate preference of the single output cell considered depends on the initial conditions of the synaptic weights. A multiple output version of this model with $\mathbf{B} = \mathbf{0}$ would therefore generate a random pattern of selectivities as a function of cortical position. Ocular dominance and orientation are organized in a highly structured map across primary visual cortex. For example, ocular dominance alternates between the right and left eyes in a roughly periodic manner (figure 8.7B). Such maps can arise in the context of Hebbian development if the intracortical connection matrix $\mathbf{B}$ is chosen appropriately.

In this chapter, we analyze the development of ocular dominance maps, leaving the study of orientation maps to chapter 9. As in a previous section, we use a continuous labeling scheme, but, in this case, for cortical neurons. The two-dimensional labeling vectors $\vec{a}$ and $\vec{b}$ used previously to label receptive field locations of LGN cells are now used to label the cortical location of a simple cell. Thus, $v(\vec{a})$ is the output of a neuron located at position $\vec{a}$ on the cortical sheet. With this continuous labeling, the intracortical connection matrix $\mathbf{B}$ becomes a function $B(\vec{a} - \vec{b})$ that determines the strength of the connections from cortical neurons located at position $\vec{b}$ to neurons located at $\vec{a}$. In writing B as a function of $\vec{a} - \vec{b}$, we have assumed translation invariance, which means that connections between neurons separated by a given displacement are independent of the

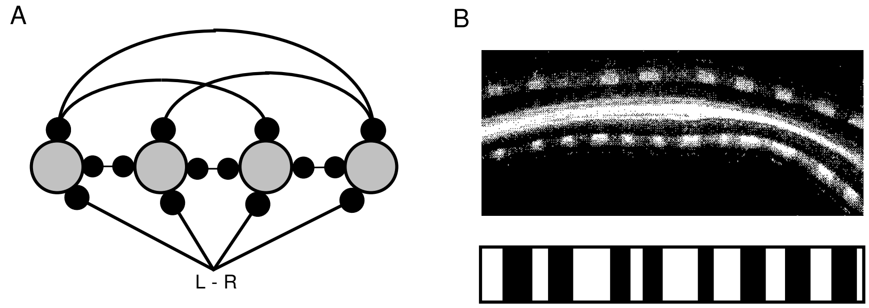

Figure 8.7: The development of ocular dominance in a Hebbian model. A) The simplified model in which the difference between left- and right-eye inputs from a single retinal location is fed to an array of cortical neurons. B) Ocular dominance maps. The upper panel shows an audioradiogram of cat primary visual cortex radioactively labeled to distinguish regions activated by one eye or the other. The alternating light and dark areas along the cortical regions at the top and bottom show alternate left- and right-eye innervation. The central region is white matter where fibers are not segregated by ocular dominance. The lower panel is a sample pattern of innervation after Hebbian development for 512 cortical units. Black and white regions denote units dominated by right- and left-eye projections respectively. (B data of S. LeVay adapted from Nicholls *et al*, 1992)

absolute location of the neurons within the region of cortex we are considering. Similarly the matrix $\mathbf{K}$ is replaced by the function $K(\vec{a}-\vec{b})$ which is the continuous analog of the matrix inverse $(\mathbf{I}-\mathbf{B})^{-1}$.

To simplify the model, we consider a population of cortical neurons innervated by LGN cells with receptive fields centered at a single retinal location (figure 8.7). We saw in the analysis of ocular dominance development for a single cortical unit that the relevant combination for Hebbian development with a subtractive constraint was the difference between left- and right-eye inputs. For this reason, we let the input activity variable u represent the difference between left- and right-eye LGN firing rates and interpret the feedforward weight to cortical neurons at location $\vec{a}$, $w(\vec{a})$, as the total strength of left-eye input minus the strength of right-eye input. Because left- and right-eye inputs have been combined and only a single receptive field location is considered, there is only one input variable u in this model, and the correlation $\langle uu\rangle$ is a number Q rather than a matrix.

Taking into account the continuous labeling, which turns sums into integrals, and the simplified structure of the feedforward input, the averaged Hebbian learning rule, equation 8.29, is in this case

$$w(\vec{a}) \rightarrow w(\vec{a}) + \epsilon\rho Q \int d\vec{b}\, K(\vec{a}-\vec{b})w(\vec{b})\,. \tag{8.30}$$

We have, once again, included a density factor ρ when replacing the sum over cortical neurons with an integral over the locations on the cortex.

To proceed further, we need to specify the form of the function K appearing in 8.30. It is assumed to be a difference of Gaussians similar in form to the Q function shown in figure 8.4B. This choice corresponds to short-range excitation and longer-range inhibition between cortical units. If $K(\vec{a} - \vec{b})$ is given by a difference of two Gaussian functions, equation 8.30 is mathematically identical to equation 8.22 with K playing the role that Q did in 8.22. The fastest growing modes for equation 8.30 are thus cosines, as in equation 8.24,

$$\mathbf{e}(\vec{a}) = \cos(\vec{k} \cdot \vec{a} - \phi) \tag{8.31}$$

with $|\vec{k}|$ determined by the peak of the Fourier transform of $K(\vec{a} - \vec{b})$. If constraints do not interfere too severely, the ultimate weight function $W(\vec{a})$ after Hebbian development will be proportional to a linear combination of cosine functions.

The sign oscillations of these cosine functions generate an alternating pattern of left- and right-eye innervation resembling the ocular dominance maps seen in primary visual cortex (upper panel figure 8.7B). The width of the resulting ocular dominance stripes is determined by the magnitude of $|\vec{k}|$. The lower panel of figure 8.7B shows the result of a simulation of Hebbian development of an ocular dominance map for a one-dimensional line across cortex consisting of 512 units. In this simulation, constraints that limit the growth of synaptic weights have been included, but these do not dramatically alter the conclusions of our simplified analysis.

Causal Hebbian Learning and Prediction

We have studied learning rules that strengthen synapses when the pre- and postsynaptic neurons are simultaneously active. Hebb's rule, as originally formulated, required not merely simultaneous activity, but a causal relationship. Synapses should be strengthened when the activity of the presynaptic neuron helps *cause* the postsynaptic neuron to fire. As we saw in figure 8.1, experimentally measured LTP appears to incorporate such a causal element because induction of LTP requires that presynaptic activity precedes postsynaptic depolarization. Depolarization of the postsynaptic neuron prior to presynaptic activity leads to LTD. A full curve of this effects is seen in the data shown in figure 8.8A.

Incorporating experimental results such as those in figures 8.1 and 8.8A into a model requires introducing spikes and basing changes of synaptic strength on the relative timing of pre- and postsynaptic spikes. However, an approximate model can be constructed on the basis of firing rates. The effect of pre- and postsynaptic timing can be included in a synaptic modification rule such as 8.1 by including a temporal difference t' between the times when the firing rates of the pre- and postsynaptic neurons are evaluated. A function $H(t')$ determines the rate of synaptic modification that occurs due to postsynaptic activity occurring a time t' after presynaptic activity. The total rate of synaptic modification is determined by integrating

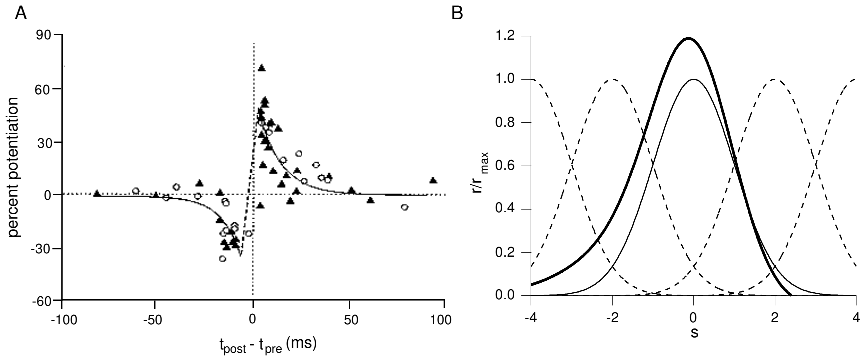

Figure 8.8: A) LTP and LTD of retinotectal synapses recorded *in vivo* in *Xenopus* tadpoles. The percent change in synaptic strength produced by multiple pairs of action potentials is plotted as a function of the time difference between post- and presynaptic firing. The filled symbols correspond to extracellular stimulation of the postsynaptic neuron and the open symbols to intracellular stimulation. The H function in equation 8.32 is proportional to the solid curve. B) Shift in a neuronal firing-rate tuning curve caused by repeated exposure to a time-dependent stimulus during training. The dashed curves and thin solid curve indicate the initial response tuning curves of a network of interconnected neurons. The thick solid curve is the response tuning curve of the neuron that initially had the thin solid tuning curve after a training period involving a time-dependent stimulus. The tuning curve increased in amplitude, broadened, and shifted as a result of temporally asymmetric Hebbian learning. The shift shown corresponds to a training stimulus with a positive rate of change, that is, one that moved rightward on this plot as a function of time. The corresponding shift in the tuning curve is to the left. (A adapted from Zhang *et al*, 1998)

over all time differences t'. If we assume that the rate of synaptic modification is bilinear in the pre- and postsynaptic rates (as it is for a Hebbian rule), we find that the rate of change of a synaptic weight w at time t is given by

$$\tau \frac{dw(t)}{dt} = \int dt' \, H(t') r_{\text{post}}(t+t') r_{\text{pre}}(t) \, . \qquad (8.32)$$

The solid curve in figure 8.8A is an example of an H function. A critical feature of the H functions implied by the data is that they are asymmetric functions of time.

The temporal asymmetry of H has a dramatic effect on synaptic weight changes because it causes them to depend on the temporal order of the activity generated during training. This allows synaptic weights to store information about temporal sequences. To see this, consider a network with the architecture of figure 8.6. Suppose that, before training, the average response in output unit a is equal to the firing rate tuning curve $f_a(s)$, and that the different tuning curves peak at different stimulus values as depicted by the dashed and thin solid curves in figure 8.8B. The tuning curve $f_a(s)$ reaches its maximum at a stimulus value $s = s_a$.

We now examine what happens when the learning rule 8.32 is applied throughout a training period during which the stimulus being presented is an increasing function of time. Such as stimulus excites the different neurons in the network sequentially. For example, the neuron with $s_a = -2$ is active before the neuron with $s_a = 0$, which in turn is active before the neuron with $s_a = 2$. If the variation of the stimulus is rapid enough, the interval between the firing of the neuron with $s_a = -2$ and that with $s_a = 0$ will fall within the window for LTP depicted in figure 8.8A. This means that a synapse from the neuron with $s_a = -2$ to the $s_a = 0$ neuron will be potentiated. On the other hand, because the neuron with $s_a = 2$ fires after the $s_a = 0$ neuron, a synapse from it to the $s_a = 0$ neuron will be depressed by the temporally asymmetric learning rule.

The effect of this type of learning on the tuning curve in the middle of the array (the thin solid curve in figure 8.8B) is shown by the thick solid curve in figure 8.8B. After the training period, the neuron with $s_a = 0$ receives strengthened input from the $s_a = -2$ neuron and weakened input from the neuron with $s_a = 2$. This broadens and shifts the tuning curve of the neuron with $s_a = 0$ to lower stimulus values. The leftward shift seen in figure 8.8B is a result of the temporal character of the learning rule and the temporal evolution of the stimulus during training. Note that the shift is in the direction opposite to the motion of the stimulus during training. This backward shift has an interesting interpretation. If the same time-dependent stimulus is presented again after training, the neuron with $s_a = 0$ will respond earlier than it did prior to training. The training experience thus causes the neuron to develop a response that predicts the behavior of the training stimulus. We have selected the neuron with $s_a = 0$ as a representative example. The same shift occurs for the response tuning curves of all the neurons excited by the time-dependent stimulus during training.

Enlargements and backward shifts of neural response tuning curves similar to those predicted from the temporal asymmetry of LTP and LTD induction have been seen in recordings of hippocampal place cells in rats (Mehta, Barnes, and McNaughton, 1997). Figure 8.9A plots the average width and location of place fields recorded while a rat ran repeated laps around a closed track. Over time, the place field grew in size and shifted backward along the track relative to the direction the rat moved.

The predictive nature of the neural representation of the stimulus following training can be seen particularly clearly if we use the population decoding approach discussed in chapter 3. This is illustrated in figure 8.9B, which shows the effect of training on the stimulus location encoded by the activity of a network of neurons with an array of two-dimensional Gaussian tuning curves. The synaptic modification rule 8.32 was used during training. Originally, before training, neuron a in the network had a tuning curve centered at the location given by the two-dimensional vector $\vec{s}_a$. Before and after training, the encoded stimulus location was determined from the output activity as $\sum r_a \vec{s}_a / \sum r_a$ (see chapter 3). Note that the same

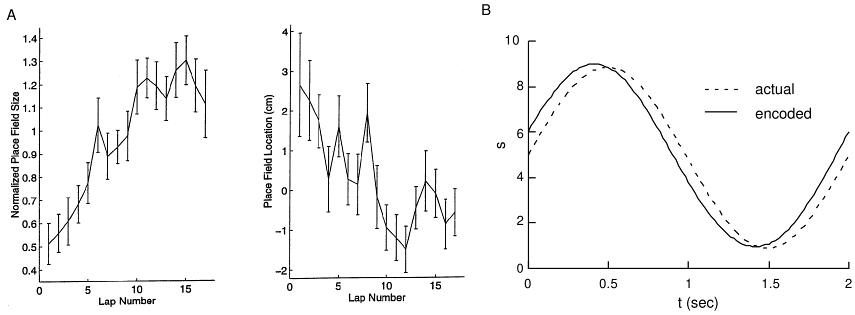

Figure 8.9: A) Enlargement and shift of place fields. Average size (left figure) and location (right figure) of place fields measured while a rat traversed laps around a closed track. Over sequential laps, the place fields expanded and shifted backward relative to the direction the rat moved. B) Comparison of actual and coded values after training. Two-dimensional Gaussian tuning curves were arranged uniformly over a grid. During training, the rule 8.32 was applied while the stimulus traversed a circle within this grid. After training the same stimulus was used as input to the network, and the resulting activity was decoded to estimate the stimulus location that it encoded. The x-coordinate of the actual stimulus location is plotted as the dashed curve, and the x-coordinate of the corresponding encoded location is indicated by a solid curve. The encoded location leads and predicts the actual location. (A from Mehta, Barnes, and McNaughton 1998; B adapted from Abbott and Blum, 1996)

preferred location $\vec{s}_a$ was used before and after training. In figure 8.9B, the dashed line shows the x-coordinate of the actual stimulus location as a function of time, and the solid line is the corresponding encoded location. The output of the network, as decoded by this scheme, is shifted in time so that it provides a prediction of where the stimulus will be in about 50 ms.

In another application of temporally asymmetric Hebbian learning, Gerstner, Kempter, van Hemmen, and Wagner (1996) showed that a learning rule with a highly temporally compressed H function (nonzero only in the millisecond range around zero) could result in neurons highly selective for precise spike timing. They applied this rule to model neurons in the auditory system of the barn owl which can respond selectively to spikes coming from the two ears separated by submillisecond intervals.

Trace Learning

Integrating equation 8.32 from $t = 0$ to a final time $t = T$, and assuming that $w = 0$ initially, we can rewrite the temporal learning rule as

$$w(T) = \frac{1}{\tau}\int_0^T dt\, r_{\text{pre}}(t)\int dt'\, H(t')r_{\text{post}}(t+t')\,. \tag{8.33}$$

This equation shows that temporal Hebbian learning depends on the correlation between the presynaptic activity and a temporally filtered version of the postsynaptic activity. Földiák (1991) suggested that this rule (with a suitably chosen $H(t')$) is appropriate for modeling the development of invariant responses. For instance, neurons in infero-temporal cortex can respond selectively to particular objects independent of their location within a wide receptive field. The idea is that objects persist in the visual environment for characteristic lengths of time, but meanwhile move across the retina. If the learning rule in equation 8.33 filters postsynaptic activity at this characteristic time scale, it can strengthen the synapses from the presynaptic units which are active for all the positions on the retina adopted by the object while it persists. As a result, the response of the postsynaptic cell comes to be independent of the position of the object on the retina. Wallis *et al* (1993) and Wallis and Baddeley (1997) have shown that such a scheme can give rise to invariant responses. This form of learning is called trace learning because of its dependence on the past history or trace of the stimulus.

The BCM Rule

For biological applications, it is more appropriate to write a synaptic modification rule in terms of the firing rates r_{pre} and r_{post} rather than the activities u and v. A simple Hebbian form for the function F in equation 8.1, $F = r_{\text{pre}} r_{\text{post}}$, only allows synapses to increase in strength because firing rates cannot be negative. We consider two modifications of this basic Hebbian rule; one that allows for both increases and decreases of synaptic weights, and another that incorporates the dependence of synaptic plasticity on the timing of pre- and postsynaptic activity.

Experiments indicate that synapses can depress in strength if presynaptic activity is accompanied by low levels of postsynaptic activity. High levels of postsynaptic activity, on the other hand, tend to produce potentiation. These results are incorporated into an alternate synaptic plasticity rule, the BCM rule (Bienenstock, Cooper, and Munro, 1982), which sets

$$\tau \frac{dw}{dt} = r_{\text{post}} r_{\text{pre}} \left(r_{\text{post}} - \theta \right) . \tag{8.34}$$

The variable θ defines a threshold that determines whether depression ($dw/dt < 0$) or potentiation ($dw/dt > 0$) occurs when both the pre- and postsynaptic neurons fire. Depression occurs for $r_{\text{post}} < \theta$ and potentiation for $r_{\text{post}} > \theta$.

If the threshold θ is held fixed, the BCM rule, like standard Hebbian rules, produces unbounded weight growth. Synaptic modifications due to the rule 8.34 can be stabilized against unbounded growth by allowing the threshold θ to slide. The critical condition is that θ must grow more rapidly than r_{post} for large r_{post} so that an overly active postsynaptic neuron will induce depression. In addition to the above equation describing modification of the synaptic weight, the BCM rule specifies that

$$\tau_\theta \frac{d\theta}{dt} = \left(r_{\text{post}}^2 - \theta\right) \tag{8.35}$$

where τ_θ sets the time scale for modification of the threshold. Adaptation of the threshold is assumed to be relatively slow, so that θ is approximately equal to a running time average of r_{post}^2.

With its sliding threshold, the BCM rule does not require additional saturation and normalization constraints. Because strong weights for any of the inputs affect the threshold, the BCM rule implements competition between inputs. It can be used to model activity-dependent development of cortical selectivities, such as ocular dominance, without requiring subtractive normalization. The BCM rule can be related to a statistical algorithm called projection pursuit. Analysis of the BCM rule along these lines is presented in chapter 11.

Principal Component Analysis

There is an interesting relationship between Hebbian learning for a single linear feedforward-driven unit (equation 8.3) and a technique of data reduction called principal component analysis. To reveal this relationship, we will ignore constraints and saturation bounds on synaptic weights and study the effects of the Hebb rule acting alone. In addition, we will not restrict the signs of either the input or output activities $\mathbf{u}$ and v. We originally interpreted these activities as representing pre- and postsynaptic firing rates, but these cannot be negative. An activity variable that takes both positive and negative values can be interpreted as the differences between a firing rate and a fixed background rate, or between the firing rates of two neurons being treated as a single unit. A Hebbian rule described by equations 8.1 or 8.2 with $F = vu$ in this case involves a nontrivial extension of Hebb's original idea. When u and v are both positive, $F = vu > 0$ producing the type of synaptic strengthening envisioned by Hebb. When u and v have opposite signs, $F < 0$ and the rule incorporates depression of synaptic strength due to anti-correlation of pre- and postsynaptic activities. If u and v are both negative, the rule produces a distinctly non-biophysical form of synaptic strengthening. Nevertheless, it is instructive to study its implications.

Figure 8.10A provides a geometric picture of Hebbian learning applied to a unit described by equation 8.3 with two inputs ($N = 2$). In this example, the inputs used during the training period were chosen from a two-dimensional Gaussian distribution with a larger variance in one direction than in the other. This resulted in the elliptical distribution of points seen in the figure. The initial weight vector $\mathbf{w}(0)$ was chosen randomly. After the Hebbian learning process, the two-dimensional weight vector was

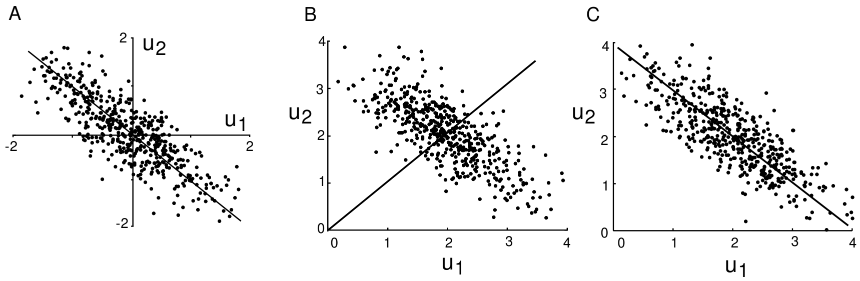

Figure 8.10: Hebbian learning and principal component analysis. A) An example of unsupervised and unconstrained Hebbian learning. The filled circles show the inputs $\mathbf{u} = (u_1, u_2)$ used during a training period while a Hebbian learning rule was applied. These input points were generated randomly from a two-dimensional Gaussian distribution with unequal variances in the two diagonal directions. After training, the vector of synaptic weights $\mathbf{w} = (w_1, w_2)$ was aligned parallel to the solid line. B) Correlation-based learning with non-zero mean input. Input vectors were generated as in A except that the distribution was shifted to produce an average value $\overline{\mathbf{u}} = (2, 2)$. After a training period during which a Hebbian learning rule was applied, the synaptic weight vector was aligned parallel to the solid line. Note that correlation-based Hebbian learning does not align the weight vector along the axis of maximum variance in this case. C) Covariance-based learning. Points from the same distribution as in B were used while a covariance-based Hebbian rule was applied. The weight vector now aligns parallel to the solid line which falls along the axis of maximum variance.

proportional to the principal eigenvector of the input correlation matrix and was aligned parallel to the line in figure 8.10A. This is the direction along which the cloud of input points has the largest variance.

Principal component analysis is a technique for generating an approximate description of a set of N-dimensional data points using less than N coordinates. Any N-dimensional vector $\mathbf{u}$ can be expressed in a Cartesian coordinate system as

$$\mathbf{u} = \sum_{\mu=1}^{N} (\mathbf{u} \cdot \mathbf{g}_\mu)\mathbf{g}_\mu \tag{8.36}$$

where the N basis vectors $\mathbf{g}_\mu$ are orthogonal and normalized to one so that $\mathbf{g}_\mu \cdot \mathbf{g}_\nu = \delta_{\mu\nu}$. The vector $\mathbf{u}$ is completely specified by the N coordinates $\mathbf{u} \cdot \mathbf{g}_\mu$ for $\mu = 1, 2, \ldots, N$. Unlike this exact representation in terms of N coordinate values, principal component analysis attempts to describe N-dimensional vectors approximately using $M < N$ coordinates. The approximate expression for $\mathbf{u}$ looks just like equation 8.36 except that it only has M terms,

$$\mathbf{u}_{\text{approx}} = \sum_{\mu=1}^{M} (\mathbf{u} \cdot \mathbf{g}_\mu)\mathbf{g}_\mu \, . \tag{8.37}$$

Principal component analysis is a method for determining the basis vectors $\mathbf{g}_\mu$, with $\mu = 1, 2, \ldots, M$, that provide the most accurate approximation for a set of vectors that we wish to describe in this reduced way. The optimal basis is found by minimizing the averaged squared difference between the approximate and actual vectors being described, $E = \langle |\mathbf{u} - \mathbf{u}_{\text{approx}}|^2 \rangle$, subject to the constraint that the basis vectors are orthogonal and of unit length. The brackets here denote an average over all of the $\mathbf{u}$ vectors in the set we wish to describe. As we will see, it is sufficient to impose only the constraint of unit length by the method of Lagrange multipliers. Substituting expression 8.37 for $\mathbf{u}_{\text{approx}}$, we find that we must minimize

$$\langle |\mathbf{u} - \sum_{\mu=1}^{M} (\mathbf{u} \cdot \mathbf{g}_\mu)\mathbf{g}_\mu|^2 \rangle - \sum_{\mu=1}^{M} \lambda_\mu \mathbf{g}_\mu \cdot \mathbf{g}_\mu \, . \tag{8.38}$$

In this expression, we have introduced a Lagrange multiplier λ_μ for each basis vector, but have defined these, for later convenience, with a minus sign relative to the normal convention. The result of setting the derivative of this expression with respect to $\mathbf{g}_\mu$ to zero is

$$\langle \mathbf{u}(\mathbf{u} \cdot \mathbf{g}_\mu) \rangle = \mathbf{Q} \cdot \mathbf{g}_\mu = \lambda_\mu \mathbf{g}_\mu \tag{8.39}$$

where we have used the definition of the correlation matrix, equation 8.6. This equation is identical to the eigenvalue equation 8.10 with the Lagrange multiplier playing the role of the eigenvalue, so we find that the basis vectors $\mathbf{g}_\mu$ are equal to M of the N eigenvectors $\mathbf{e}_\mu$. The remaining question is which M of the N possible eigenvectors should be chosen? To answer this question, we substitute the result 8.39 into the expression for the average squared error to obtain

$$E = \mathrm{Tr}\,\mathbf{Q} - \sum_{\mu=1}^{M} \lambda_\mu \tag{8.40}$$

where $\mathrm{Tr}\,\mathbf{Q}$ stands for the trace operation (the sum of the diagonal elements of $\mathbf{Q}$). To minimize this error, we must make the second term, that enters with a minus sign, as large as possible. This can be done by choosing the vectors $\mathbf{g}_\mu$ to be the M eigenvectors of the correlation matrix with the biggest eigenvalues (the orthogonality condition on the vectors $\mathbf{g}_\mu$ forbids us from using the same eigenvector more than once).

A single unit obeying equation 8.3 characterizes the state of its N inputs by a single number v which is equal to the projection of $\mathbf{u}$ onto the weight vector $\mathbf{w}$. It thus corresponds to the extreme case $M = 1$ in the data reduction process we have been describing. Principal component analysis instructs us to set the single projection axis $\mathbf{w}$ to the eigenvector of the correlation matrix with the largest eigenvalue. Up to a proportionality constant, this is exactly what unsupervised Hebbian learning does.

If the average value of all the input vectors presented is zero, $\langle \mathbf{u} \rangle = 0$, the choice $\mathbf{w} = \mathbf{e}_1$ also identifies the direction in the space of inputs with the maximum variance. To see this, write $v = \mathbf{g} \cdot \mathbf{u}$ with the single $\mathbf{g}$ vector restricted to unit length, $\mathbf{g} \cdot \mathbf{g} = 1$. To maximize the variance of v subject to this constraint, we must maximize $\langle (\mathbf{g} \cdot \mathbf{u})^2 \rangle - \lambda \mathbf{g} \cdot \mathbf{g}$. The maximization proceeds exactly as above, and we find that $\mathbf{g}$ must be an eigenvector of $\mathbf{Q}$. The variance of v is then given by $\langle v^2 \rangle = \lambda$, so the maximum variance is achieved for $\mathbf{g} = \mathbf{e}_1$.

In chapter 4, we showed that the entropy of a Gaussian distributed random variable with variance σ^2 grows with increasing variance as $\log_2 \sigma$. On the basis of this result, Linsker (1988) has shown that when the input statistics and noise are Gaussian, maximizing the variance of v by the Hebb rule also maximizes the amount of information v carries about $\mathbf{u}$.

Covariance-Based Rules

The connection between Hebbian learning and principal component analysis that we have presented relies on the assumption that the average value of the input vector is zero, and this may not always be the case. For example, if firing rates rather than activities are used to characterize the inputs, their average values will not be zero unless the inputs are completely silent. Figure 8.10B illustrates what happens when $\langle \mathbf{u} \rangle = \overline{\mathbf{u}} \neq 0$ (in contrast to figure 8.10A, where $\overline{\mathbf{u}} = 0$). Correlation-based Hebbian learning now aligns the weight vector along a line from the origin to the point $\overline{\mathbf{u}}$. This is clearly not the direction of maximum variance in the input data set, so the interpretation of Hebbian learning discussed in the last section is lost.

Equation 8.8 finds the principal eigenvector of the correlation matrix $\mathbf{Q}$. If the means of the inputs are non-zero, implementing PCA requires finding the principal eigenvector of the covariance matrix $C_{ab} = \langle (u_a - \overline{u}_a)(u_b - \overline{u}_b) \rangle = Q_{ab} - \overline{u}_a \overline{u}_b$ rather than the correlation matrix. To accomplish this, the averaged learning rule should be

$$\mathbf{w} \to \mathbf{w} + \epsilon \mathbf{C} \cdot \mathbf{w} \, . \tag{8.41}$$

This can be achieved in several ways: the homosynaptic rule

$$\mathbf{w} \to \mathbf{w} + \epsilon (v - \overline{v}) \mathbf{u} \, , \tag{8.42}$$

the heterosynaptic rule

$$\mathbf{w} \to \mathbf{w} + \epsilon v (\mathbf{u} - \overline{\mathbf{u}}) \, , \tag{8.43}$$

or the full covariance rule

$$\mathbf{w} \to \mathbf{w} + \epsilon (v - \overline{v})(\mathbf{u} - \overline{\mathbf{u}}) \, , \tag{8.44}$$

where $\overline{v} = \langle v \rangle = \mathbf{w} \cdot \overline{\mathbf{u}}$. In these rules, the change in synaptic weight is determined by the difference between the actual and average value of the output, input, or both. All of the rules lead to 8.41 when averaged over input vectors. Figure 8.10C shows that when such correlation-based rules are applied to an input set with nonzero mean, the weight vector aligns with the axis of maximum variance. In such cases, the weight vector aligns with the principal eigenvector of the covariance matrix rather than the correlation matrix.

Anti-Hebbian Learning and Redundancy Reduction

When many output units share the same inputs, they are likely to develop similar responses unless some mechanism actively reduces redundancy. For example, if Hebbian learning is applied independently to each output unit, they will all represent the same principal component of the input data. Fixed connections between the output units, as considered above, do not typically fix this problem adequately. As discussed in chapter 4, a natural first step towards eliminating redundancy is to decorrelate the outputs. An anti-Hebbian learning rule, presented below, achieves output decorrelation by making

$$\langle \mathbf{v}\mathbf{v} \rangle = \mathbf{I} \quad \text{or equivalently} \quad \langle v_a v_b \rangle = \delta_{ab} \,. \tag{8.45}$$

This corresponds to decorrelation because the off-diagonal elements of $\langle \mathbf{v}\mathbf{v} \rangle$ are zero. The outputs are also whitened, in the sense discussed in chapter 4, because all of the elements of $\mathbf{v}$ have the same variance.

As the name implies, anti-Hebbian learning causes synapses to decrease (rather than increase) in strength when there is simultaneous pre- and postsynaptic activity. This is accomplished simply by changing the sign of the modification in equation 8.4. Anti-Hebbian learning is believed to be the predominant form of plasticity from parallel fibers to Purkinje cells in the cerebellum, although this may be a special case because Purkinje cells inhibit rather than excite their targets. Hebbian learning makes weights grow without bound, so it is often combined with some form of weight decay, as we have seen. Anti-Hebbian learning has the opposite problem of making weights tend toward zero, so it needs to be combined with some form of weight growth.

Goodall (1960) proposed a method of decorrelating neurons driven through fixed feedforward weights $\mathbf{W}$ by modifying the recurrent weights $\mathbf{B}$ between them. The recurrent weights develop according to the anti-Hebbian rule

$$\mathbf{B} \rightarrow \mathbf{B} + \epsilon(-(\mathbf{W} \cdot \mathbf{u})\mathbf{v} + \mathbf{I} - \mathbf{B}) \,. \tag{8.46}$$

The minus sign in the term $-(\mathbf{W} \cdot \mathbf{u})\mathbf{v}$ embodies anti-Hebbian learning. This term is non-local, because the change in the weight of a given synapse depends on the total feedforward input to the postsynaptic neuron, not merely on the input at that particular synapse. The term $\mathbf{I} - \mathbf{B}$ prevents the weights from tending to zero by forcing them toward $\mathbf{I}$. This rule requires the existence of autapses, synapses that a neuron makes onto itself.

If the Goodall learning rule converges to the identity map $\mathbf{B} \rightarrow \mathbf{B}$ (which we will not prove), the term proportional to ϵ in 8.46 must vanish, on average, so

$$\langle(\mathbf{W}\cdot\mathbf{u})\mathbf{v}\rangle = \mathbf{I} - \mathbf{B} = \mathbf{K}^{-1}\,. \tag{8.47}$$

Multiplying both sides by $\mathbf{K}$ we find

$$\langle(\mathbf{K}\cdot\mathbf{W}\cdot\mathbf{u})\mathbf{v}\rangle = \langle\mathbf{v}\mathbf{v}\rangle = \mathbf{I}\,, \tag{8.48}$$

and thus the outputs are decorrelated. This algorithm can be used to implement the decorrelation and whitening of optimal filters discussed in chapter 4.

Hebbian synaptic modification applied to a single linear unit generates a one-dimensional principal component representation. With M output units, it is possible to generate principal component representations with $0 < M < N$ basis vectors provided that some form of competition or decorrelation is included to prevent redundancy. This can be done in a feedforward network at the level of the synaptic learning rule, and a number of such schemes have been proposed (Williams, 1985; Oja & Karhunen, 1985; Oja, 1989, Sanger, 1989). Alternatively, in a network with recurrent connections, an Oja-type Hebbian learning rule can be used for the feedforward weights and anti-Hebbian learning can be applied to the recurrent weights to produce outputs that provide a multi-dimensional principal component representation (Foldiak, 1989; Leen, 1991; Plumbley, 1991).

8.3 Supervised Hebbian Learning

In unsupervised learning, inputs are imposed on the system during the training period, but the output is determined by the network dynamics using the current values of the weights. In supervised learning, both a set of inputs and the corresponding 'desired' outputs are imposed during training. Consider a unit connected to N inputs and suppose that the training set of inputs consists of P vectors $\mathbf{u}_m$ with $m = 1, 2, \ldots, P$. The desired outputs associated with these inputs are labeled v_m with m extending over the same range. A Hebbian learning rule applied during this supervised learning process produces the change

$$\mathbf{w} \rightarrow \mathbf{w} + \epsilon v_m \mathbf{u}_m \tag{8.49}$$

every time the combination $\mathbf{u}_m$, v_m is presented. If the weights are set to zero initially , the weight vector after the presentation of input-output pairs for each m value will be $\mathbf{w} = \epsilon \sum_m v_m \mathbf{u}_m$. Repeated presentations of the set of input-output pairs will merely change the proportionality factor in this equation. Since this factor does not affect the results discussed below, we will set it to $1/N$ for convenience, and describe the result of supervised Hebbian learning simply as

$$\mathbf{w} = \frac{1}{N}\sum_{m=1}^{P} v_m \mathbf{u}_m \,. \tag{8.50}$$

We will use this result to study a single unit learning binary input-output maps and to analyze a network that exhibits associative memory. Before presenting these examples, we consider another supervised learning task, function approximation by neural networks.

Function Approximation

In chapter 1, we studied populations of neurons (labeled by an index $a = 1, 2, \ldots, N$) that responded to a stimulus value s, by firing at average rates $f_a(s)$. The individual outputs of these neurons represent different functions of the stimulus parameter s. However, a function $h(s)$ that is not equal to any of the single neuron tuning curves must be represented by combining the responses of a number of units. This can be done using the network shown in figure 8.11. The average activity level of the output unit in this network, in response to stimulus value s, is given by a linear summation rule,

$$v(s) = \sum_{a=1}^{N} w_a f_a(s) = \mathbf{w}\cdot\mathbf{f}(s) \tag{8.51}$$

where $\mathbf{w}$ and $\mathbf{f}$ are vectors with components w_a and f_a respectively.

We seek to determine conditions and learning rules that set $v(s) \approx h(s)$ so that the activity of the output unit represents the function to a desired degree of accuracy. Unless the set of output functions being represented is highly restricted, perfect function representation, $v(s) = h(s)$, requires an infinite number of neurons. We will consider instead function approximation, $v(s) \approx h(s)$, using a finite number of units. We will strive to choose the synaptic weights so that the approximation is as accurate as possible.

In equation 8.51, the input tuning curves $\mathbf{f}(s)$ act as a basis for representing the output function $h(s)$, and for this reason they are called basis functions. Without additional conditions, the set of basis functions needed to represent a set of output functions is not unique. We discuss optimal choices of basis functions using a number of different criteria in chapter 11. A set of basis functions that can represent any member of a class of functions (with a specified measure of accuracy) using a linear sum as in equation 8.51 is called *complete* for this class. For the sets of complete functions typically used in mathematics, such as the sines and cosines used in a Fourier series, the weights in equation 8.51 for any given target function are unique. When neural tuning curves are used to expand a function, the weights

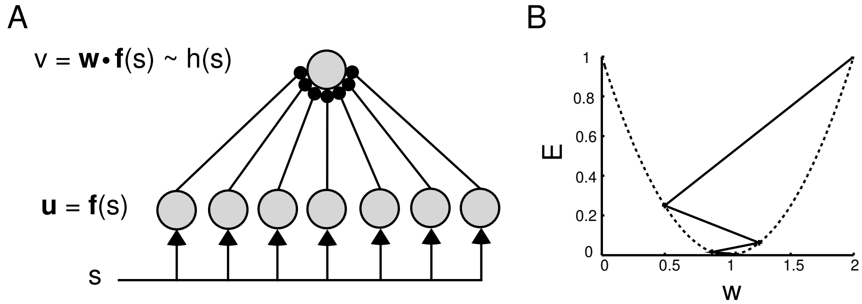

Figure 8.11: A) Network for representing functions. The value of an input variable s is encoded by the activity of a population of neurons with tuning curves $f_a(s)$, for $a = 1, 2, \ldots, N$. This activity drives an output neuron through a set of weights w_a to create an average firing rate v equal to the function $h(s)$. B) Gradient descent. The dashed curve shows $E(w) = (w-1)^2$; the jagged straight lines join successive values of w, starting from $w{=}2$ and using the gradient descent rule $w \to w{-}1.5(w{-}1)$. The value of $\epsilon{=}1.5$ used here is large for illustrative purposes.

tend not to be unique and the set of input functions is called overcomplete.

Suppose that the function-representation network of figure 8.11A is provided a sequence of P sample stimuli s_m for $m = 1, 2, \ldots, P$ and the corresponding function values $h(s_m)$ during a training period. To make $v(s_m)$ match $h(s_m)$ as closely as possible for all m, we minimize the total squared difference between them

$$E(\mathbf{w}) = \sum_{m=1}^{P} [h(s_m) - v(s_m)]^2 = \sum_{m=1}^{P} \left[h(s_m) - \sum_{a=1}^{N} w_a f_a(s_m) \right]^2 . \tag{8.52}$$

Equations for the weights that minimize this error, called the normal equations, are obtained by setting the derivative of the error with respect to the weights to zero, yielding the condition

$$\sum_{m=1}^{P} \sum_{b=1}^{N} f_a(s_m) f_b(s_m) w_b = \sum_{m=1}^{P} f_a(s_m) h(s_m) . \tag{8.53}$$

A learning rule that constructs weights that represent a function over the sample points to the highest degree of accuracy should generate a set of weights that satisfy the normal equations.

Gradient Descent Learning

Up to this point, our discussion of the relationship between synaptic plasticity and learning has been based on determining what tasks a given synaptic modification rule allows a network to perform. Another approach, appropriate for function approximation, is to start with a task, and derive appropriate synaptic modification rules. If performance of the task is quantified by an error or cost function that depends on the synaptic weights, the learning rule can be derived as a procedure for minimizing this function. Gradient descent learning is the most straightforward instantiation of this approach.

Gradient descent is an iterative procedure for finding the minimum of a function such as $E(\mathbf{w})$ of equation 8.52. The procedure starts with an initial guess about the location of the minimum, and then improves this guess on each iteration by making the replacement

$$\mathbf{w} \rightarrow \mathbf{w} - \epsilon \nabla_w E(\mathbf{w}) \tag{8.54}$$

where $\nabla_w E(\mathbf{w})$ is the vector with components $\partial E / \partial w_a$. The replacement is sensible because the vector $-\nabla_w E(\mathbf{w})$ points in the direction along which E decreases most rapidly. In the limit of small ϵ and sufficiently smooth $E(\mathbf{w})$, this process is guaranteed to reduce (or at least not to increase the cost function) because, using a first order Taylor series, it replaces $E(\mathbf{w})$ by

$$E(\mathbf{w} - \epsilon \nabla_w E(\mathbf{w})) \approx E(\mathbf{w}) - \epsilon \nabla_w E(\mathbf{w}) \cdot \nabla_w E(\mathbf{w}) \leq E(\mathbf{w}) \tag{8.55}$$

since $\nabla_w E(\mathbf{w}) \cdot \nabla_w E(\mathbf{w}) \geq 0$. Figure 8.11B shows a simple example of gradient descent. If $E(\mathbf{w})$ has many minima, gradient descent will find only one of them (a local minimum), and not necessarily the one with the lowest value of $E(\mathbf{w})$ (the global minimum). Local maxima can be found using gradient ascent by reversing the sign in equation 8.54.

Often the total error $E(\mathbf{w})$ is given by the sum over many presented stimuli of a stimulus-dependent error function $E(\mathbf{w}) = \sum_m E(\mathbf{w}; s_m)$ as in equation 8.52. In this case, a procedure called stochastic gradient descent can be used. When the stimulus s_m is present, the weight vector is updated by

$$\mathbf{w} \rightarrow \mathbf{w} - \epsilon \nabla_w E(\mathbf{w}; s_m) . \tag{8.56}$$

The relation between stochastic gradient descent and gradient descent is the same as the relationship between the Hebbian learning rule 8.4 and its average over input presentations, 8.7. Stochastic gradient descent is particularly useful because it allows learning to take place continuously while sample inputs are presented during a training period. There are more efficient methods of searching for minima of functions than stochastic gradient descent, but many of them are complicated to implement.

The parameter ϵ that controls the magnitude of the weight adjustment must be small enough to validate the linear approximation in equation 8.55 and to allow stochastic gradient descent to average to true gradient descent. Getting the weights to converge to fixed values often requires reducing the value of ϵ as the minimum of E is approached. Often a fixed value of ϵ is used, in which case the weights may never completely settle down during the training period.

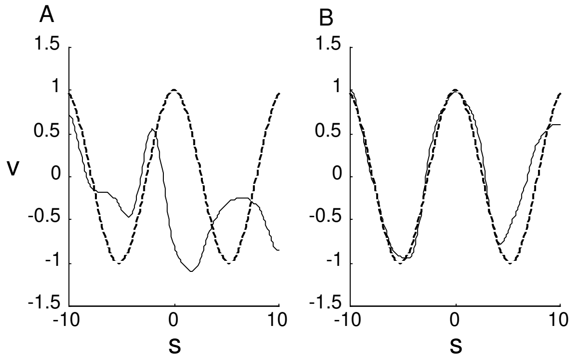

Figure 8.12: Eleven input neurons with Gaussian tuning curves drive an output neuron to approximate a sine function. The input tuning curves were $f_b(s) = \exp[-0.5(s-\xi_b)^2]$ with $\xi_b = -10, -8, -6, \ldots, 8, 10$. A delta learning rule with $\epsilon = 1$ was used to set the weights. Example points were chosen randomly in the range between -10 and 10. The firing rate of the output neuron is plotted as a solid curve and the sinusoidal target function as a dashed curve. A) The firing rate of the output neuron when random weights in the range between -1 and 1 were used. B) The output firing rate after weight modification using the delta rule for 20 sample points. C) The output firing rate after weight modification using the delta rule for 100 sample points.

The Delta Rule

The delta rule is the synaptic modification rule that results from applying stochastic gradient descent to the quadratic error function used in equation 8.52

$$E(\mathbf{w}; s_m) = (h(s_m) - \mathbf{w} \cdot \mathbf{f}(s_m))^2 \,. \tag{8.57}$$

Using equation 8.56, we find that, when the stimulus s_m and desired response $h(s_m)$ are present, the synaptic weight vector should be updated according to (if we absorbed a factor of 2 into the definition of ϵ)

$$\mathbf{w} \rightarrow \mathbf{w} + \epsilon\,(h(s_m) - v(s_m))\,\mathbf{f}(s_m) \tag{8.58}$$

with $v(s_m) = \mathbf{w} \cdot \mathbf{f}(s_m)$. The delta rule changes the weight w_a by an amount proportional to the product of the presynaptic activity $f_a(s)$ and the postsynaptic function approximation discrepancy $h(s) - v(s)$. The error function 8.57 used in this case has only one minimum, and given appropriate reductions in ϵ, the learning rule will always locate this global minimum.

Figure 8.12 shows the result of modifying an initially random set of weights using the delta rule. Ultimately, an array of input neurons with Gaussian tuning curves drives an output neuron that quite accurately represents a sine function.

The presence of the target function $h(s)$ in the delta learning rule makes it difficult to imagine how it could be realized in a biological system. Indeed, biological examples of supervised learning, in which the correct response is imposed on individual neurons, are difficult to conceive. Unsupervised and reinforcement learning (chapter 10) appear more plausible from a biological perspective.

Classification and The Perceptron

The perceptron is a nonlinear map that classifies binary inputs into one of two categories using a single binary output. Unlike the linear element considered thus far in this chapter, the perceptron restricts the input values u_a and output v to just two values, ± 1. We can think of these two states as representing units that are either active or inactive. The restriction to binary values is imposed by fiat as a condition on the inputs, and imposed on the output by modifying the linear rule of equation 8.3 to

$$v = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{u}) \tag{8.59}$$

where the function sgn is defined by

$$\mathrm{sgn}(\mathbf{w} \cdot \mathbf{u}) = \begin{cases} +1 & \text{if} \quad \mathbf{w} \cdot \mathbf{u} \geq 0 \\ -1 & \text{if} \quad \mathbf{w} \cdot \mathbf{u} < 0 \,. \end{cases} \tag{8.60}$$

The supervised learning task for the perceptron is to learn P associations between input vectors $\mathbf{u}_m$ and binary outputs v_m. How well the perceptron performs the desired task depends on the type of association being learned. The weight vector defines an $N-1$-dimensional subspace (the subspace perpendicular to $\mathbf{w}$ passing through the origin) that cuts the N-dimensional space of input vectors into two regions. For the input-output relation to be learned perfectly, all of the inputs corresponding to $v = 1$ must fall in one of the half spaces and all of those with $v = -1$ must fall in the other half (figure 8.13A shows an example), a condition called linear separability. This is a necessary and sufficient condition for the task to be exactly learnable, but does not imply that it can be learned by a Hebb rule. An instructive case to consider is when the input vectors and associated output values are chosen randomly and independently. For large N, Cover (1965) has proven that the maximum number of random associations that can be learned is $2N$, but again this does not imply that Hebbian learning can achieve this bound.

To determine how well the Hebbian weights of equation 8.50 perform, we must determine whether the output given by equation 8.59, for an input vector $\mathbf{u}_n$ chosen from the training set, is equal to the desired output v_n. Substituting 8.50 into 8.59, we find

$$v = \mathrm{sgn}\left(\frac{1}{N}\sum_{m=1}^{P} v_m \mathbf{u}_m \cdot \mathbf{u}_n\right) . \tag{8.61}$$

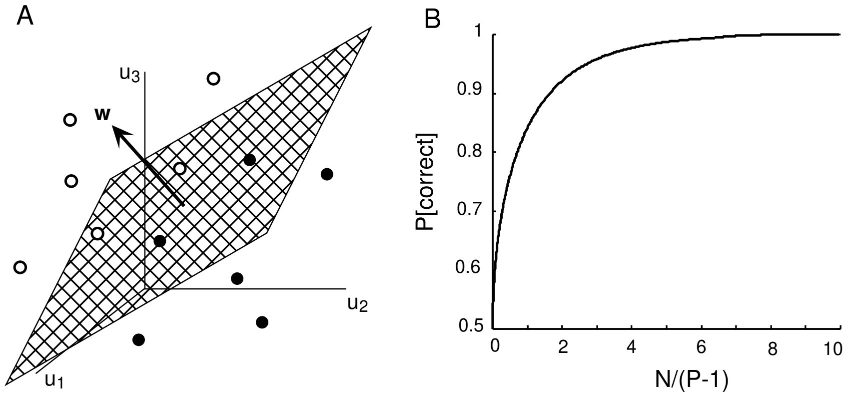

Figure 8.13: A) Separable inputs for a perceptron with three inputs. The hatched plane perpendicular to the weight vector **w** divides the three-dimensional input space into two regions. In this example, all the input points corresponding to outputs $v = 1$ (open circles) fall on one side of the plane, and all inputs corresponding to outputs with $v = -1$ (filled circles) fall on the other side of the plane so the task can be solved by a perceptron. B) Percentage of correct response for perceptron with a Hebbian weight vector learning a random binary input-output map. As the ratio of the number of inputs, N, to one less than the number of input vectors being learned, $P - 1$, grows, the percentage of correct responses goes to one. When this ratio is small, the percentage of correct responses approaches the chance level of $1/2$.

This will produce the desired result $v = v_n$ if the argument of the sgn function has the same sign as v_n. To check this, we break the sum over m in the argument of the sgn function into a single term with $m = n$ and the remaining terms with $m \neq n$,

$$\frac{1}{N}\sum_{m=1}^{P} v_m \mathbf{u}_m \cdot \mathbf{u}_n = v_n + \frac{1}{N}\sum_{m \neq n} v_m \mathbf{u}_m \cdot \mathbf{u}_n \,. \tag{8.62}$$

The term corresponding to $m = n$ is v_n because $\mathbf{u}_n \cdot \mathbf{u}_n = N$ (recall that each component of $\mathbf{u}_m = \pm 1$). Note that this term by itself gives the correct answer because $\text{sgn}(v_n)$ is equal to v_n. Thus, the perceptron algorithm will produce the correct output provided that the second term in equation 8.62 does not overwhelm the first term and change the sign of the resulting sum. The magnitude of the second term can be estimated using a probabilistic argument. The dot product $\mathbf{u}_m \cdot \mathbf{u}_n$ with $m \neq n$, for random vectors $\mathbf{u}_m$ and $\mathbf{u}_n$, is the sum of N terms that are either plus one or minus one with equal probability. For large N, the central limit theorem tells us that this sum is Gaussian distributed with zero mean and variance N. Dividing by $1/N$ we find that the sum over $m \neq n$ is the sum of $P - 1$ terms that have zero mean and variance $1/N$. Again by the central limit theorem, if P is large, this sum is Gaussian distributed with zero mean and variance $(P - 1)/N$.

If $v_n = 1$ the output of the perceptron will be correct provided that the second term in equation 8.62 is greater than -1. Similarly, if $v_n = -1$, a correct output requires the second term must be less than 1. Both cases have the same probability given by the integral of the Gaussian distribution of values over the relevant range,

$$P[\text{correct}] = \sqrt{\frac{N}{2\pi(P-1)}} \int_{-\infty}^{1} du \, \exp\left(-\frac{Nu^2}{2(P-1)}\right) \tag{8.63}$$

This result is plotted in figure 8.13B. As seen in the figure, the Hebbian perceptron performs fairly well in the large N limit, if P is less than about $0.2N$.

It is possible for the perceptron to perform considerably better than it does using a Hebbian learning rule if a non-Hebbian weight vector is used. One possibility is

$$\mathbf{w} = \sum_{m,m'=1}^{P} v_m D^{-1}_{mm'} \mathbf{u}_{m'} \tag{8.64}$$

where

$$D_{mm'} = \mathbf{u}_{m'} \cdot \mathbf{u}_m \; . \tag{8.65}$$

This construction is possible provided that the matrix with elements $D_{mm'}$ can be inverted. Invertibility requires the different input vectors to be linearly independent, which is only possible if $P \le N$. If the matrix inversion can be done, a perceptron using the weights of equation 8.64 will perform perfectly because, by construction (equation 8.65)

$$\mathbf{w} \cdot \mathbf{u}_m = \sum_{m,m'=1}^{P} v_m D^{-1}_{mm'} \mathbf{u}_{m'} \cdot \mathbf{u}_m = v_m \tag{8.66}$$

for all m values.

The delta rule of equation 8.58 is a more efficient way of generating the weights for a perceptron than the Hebb rule. The delta rule computes the change in the weight vector by comparing the desired output v_m for a given input with the actual response $\mathbf{w} \cdot \mathbf{u}_m$ and setting

$$\mathbf{w} \rightarrow \mathbf{w} + \epsilon(v_m - \mathbf{w} \cdot \mathbf{u}_m)\mathbf{u}_m \, . \tag{8.67}$$

Finally, it can be proven that repeated application of the perceptron learning rule,

$$\mathbf{w} \rightarrow \mathbf{w} + \epsilon(1 - v v_m)\mathbf{u}_m v_m \, , \tag{8.68}$$

leads to weights that solve any linearly separable binary classification problem (see for example Hertz *et al*, 1991).

Associative Memory

Associative memory is a form of memory in which a partial or approximate representation of a stored item is used to recall the full item from storage. For example we might recall every digit of a well-known phone number given a few of its digits as an initial clue. A recurrently connected, N unit version of the perceptron is capable of performing this task for memories that consist of N-bit binary strings. The network used for this purpose has the architecture of figure 8.6 with $\mathbf{W} = \mathbf{0}$ and nonzero $\mathbf{B}$. Such an arrangement is commonly called a Hopfield network.

Associative memory in a Hopfield network is based on a dynamic rule determining how the $M = N$ output values $\mathbf{v}(t)$ change over time. At any time t, the input to the output units is the vector $\mathbf{B} \cdot \mathbf{v}(t)$. At a later time $t + \Delta t$, the outputs are determined by the rule

$$\mathbf{v}(t + \Delta t) = \text{sgn}(\mathbf{B} \cdot \mathbf{v}(t)) \, . \tag{8.69}$$

The network evolves over time in discrete steps of size Δt through the repeated application of this update rule. Note that this rule restricts the outputs to the binary values ± 1. As expressed in equation 8.69, the update rule is applied to all of the different components of the vector $\mathbf{v}$ at once. This is called parallel update. An alternative is to apply the rule to one component of $\mathbf{v}$, say v_i, at a time, so that $v_i(t + \Delta t) = \text{sgn}(\mathbf{B} \cdot \mathbf{v}(t))$. In this case, the component to be updated can be chosen sequentially in a fixed order or randomly.

When this model is used for memory storage, a number of output states, denoted by $\mathbf{v}_m$ with $m = 1, 2, \ldots , P$, are defined as memory patterns. Associative recall is achieved by starting the network in an initial state that is similar, but not identical, to one of the memory patterns, $\mathbf{v}(0) \approx \mathbf{v}_m$ for some m value. In this case, approximately equal means that a significant number, but not all, of the elements of $\mathbf{v}(0)$ are the same as those of $\mathbf{v}_m$. The network then evolves through repeated application of the rule 8.69. If the recall is successful, the dynamics will converge to an unchanging fixed point equal to the memory pattern associated with the initial state, $\mathbf{v}(t + \Delta t) = \mathbf{v}(t) = \mathbf{v}_m$ for large t. Recall will fail if the network does not reach a fixed point (for example, it may cycle or evolve chaotically), or if the fixed point is not equal to the memory state $\mathbf{v}_m$.

If the weight matrix is symmetric (equal pre- $\rightarrow$ postsynaptic and post- $\rightarrow$ presynaptic coupling between all pairs of neurons) and has zero elements on its diagonal (no self-coupling), the Hopfield network will always converge to a fixed point. This can be proved by constructing the function $E = -\mathbf{v} \cdot \mathbf{B} \cdot \mathbf{v}$, called a Liapunov function. If $\mathbf{B}$ is symmetric with zero diagonal elements, the change in E produced by updating a single element v_i of the network is

$$E(t + \Delta t) = E(t) - 2\left(v_i(t + \Delta t) - v_i(t)\right) \sum_{j=1}^{N} B_{ij} v_j(t) \, . \tag{8.70}$$

Because the outputs are resticted to the values ± 1, $v_i(t+\Delta t) = -v_i(t)$ if the update changes unit i. The change in E is thus $-4v_i(t+\Delta t)\sum_j B_{ij}v_j(t)$. According to the rule 8.69, the sign of $v_i(t+\Delta t)$ is the same as the sign of $\sum_j B_{ij}v_j(t)$. Therefore, $E(t+\Delta t) < E(t)$, and E always decreases whenever any element of the network changes in accordance with the update rule 8.69. Since E is bounded, it must eventually stop decreasing, and this can only occur if all of the network elements stop changing. Because we have considered the update of a single unit, this result applies only to sequential or random, not parallel, update procedures.

The learning task for associative memory is to find a recurrent weight matrix $\mathbf{B}$ that leads to successful memory recall. All of the learning rules discussed in connection with the perceptron can be applied to this task. The generalization of the Hebbian rule to this case gives a recurrent matrix with off-diagonal elements given by

$$\mathbf{B} = \sum_{m=1}^{P} \mathbf{v}_m \mathbf{v}_m \tag{8.71}$$

and diagonal elements equal to zero. In terms of components, this means that $B_{ab} = (1-\delta_{ab})\sum_m (v_m)_a (v_m)_b$. Amit, Gutfreund and Sompolinsky (1987) found that this particular weight matrix allows an N element network to store up to $P = 0.14N$ different randomly chosen memory patterns, for large N.

The Hebb form of the weight matrix produces a relatively poor associative memory network. The same argument used to determine the capacity of the perceptron can be used to show that a Hopfield network can, in principal, store $P = 2N$ randomly chosen memory patterns. A Hebbian learning rule can be used to learn $2N$ patterns, if it is applied correctly. According to the Hebb rule, the change in the weight matrix needed to learn a memory pattern $\mathbf{v}_m$ is $\mathbf{B} \rightarrow \mathbf{B} + \epsilon \mathbf{v}_m \mathbf{v}_m$. To achieve optimal performance, the particular pattern (m value) used in this rule must not be chosen randomly; it must be selected. At any given time, the Hebb rule is only applied to the pattern with the smallest value of $\mathbf{v}_m \cdot \mathbf{B} \cdot \mathbf{v}_m$. Krauth and Mézard (1987) have proven that, in the limit $N \rightarrow \infty$, this rule can reach the limit of $P = 2N$ correctly stored input-output associations.

8.4 Further Reading and References

Further Reading

Amit, DJ (1989) *Modelling Brain Function*. New York:Cambridge University Press.

Barlow, HB (1989) Unsupervised learning. *Neural Computation* **1**:295-311.

Bishop, CM (1995) *Neural Networks for Pattern Recognition*. Oxford:Clarendon Press, Oxford University Press.

Foldiak, P (1991) Learning invariance from transformed sequences. *Neural Computation* **3**:194-200.

Gerstner, W, Kempter, R, van Hemmen, J & Wagner, H (1996) A neuronal learning rule for sub-millisecond temporal coding. *Nature* **383**:76-78.

Grossberg, S (1988) Nonlinear neural networks: Principles, mechanisms and architectures. *Neural Networks* **1**:17-61.

Hebb, DO (1949) *The Organization of Behavior: A Neuropsychological Theory*. New York:Wiley.

Hertz, JA, Palmer, RG & Krogh, A (1991) *Introduction to the Theory of Neural Computation*. New York:Addison-Wesley.

Kearns, MJ & Vazirani, UV (1994) *An Introduction to Computational Learning Theory.* Cambridge, MA: MIT Press.

Kohonen, T (1988) An introduction to neural computing. *Neural Networks* **1**:3-16.

Kohonen, T (1989) *Self-Organization and Associative Memory*. New York:Springer-Verlag.

Marr, D (1969) A theory of cerebellar cortex. *Journal of Physiology* **202**:437-470.

Marr, D (1971) Simple memory: A theory for archicortex. *Philosophical Transactions of the Royal Society of London* **B262**:23-81.

Miller, KD (1990) Correlation-based models of neural development. In MA Gluck, & DE Rumelhart, editors, *Neuroscience and Connectionist Theory*. Hillsdale, NJ:Lawrence Erlbaum, 267-354.

Miller, KD (1996) Receptive fields and maps in the visual cortex: Models of ocular dominance and orientation columns. In E Domany, JL van Hemmen & K Schulten, editors, *Models of Neural Networks, III*. New York:Springer-Verlag, 55-78.

Ripley, BD (1996) *Pattern Recognition and Neural Networks.* Cambridge, England: Cambridge University Press.

Rumelhart, DE & McClelland, JL, editors (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volumes I & II*. Cambridge, MA:MIT Press.

Tikhonov, AN & Arsenin, VY (1977) *Solutions of Ill-Posed Problems.* Washington, DC: WH Winston.

Widrow, B & Stearns, SD (1985) *Adaptive Signal Processing*. Englewood Cliffs, NJ:Prentice-Hall.

Vapnik, VN (1982) *Estimation of Dependences Based on Empirical Data.* New York, NY: Springer Verlag.

Vapnik, VN (1995) *The Nature of Statistical Learning Theory.* New York, NY: Springer.

References

Abbott, LF & Blum, KI (1996) Functional significance of long-term potentiation for sequence learning and prediction. *Cerebral Cortex* **6**:406-416.

Amit, DJ, Gutfreund, H & Sompolinsky, H (1987) Statistical mechanics of neural networks near saturation. *Annals of Physics* **173**:30-67.

Atick, JJ & Redlich, AN (1993) Convergent algorithm for sensory receptive field development. *Neural Computation* **5**:45-60.

Bienenstock, EL, Cooper, LN & Munro, PW (1982) Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience* **2**:32-48.

Cohen, M & Grossberg, S (1983) Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **9**:195-219.

Foldiak, P (1989) Adaptive network for optimal linear feature extraction. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks*. New York:IEEE Press, 401-405.

Gerstner, W & Abbott, LF (1997) Learning Navigational Maps Through Potentiation and Modulation of Hippocampal Place Cells. *Journal of Computational Neuroscience* **4**:79-94.

Girosi, F, Jones, M & Poggio, T (1995) Regularization theory and neural networks. *Neural Computation* **7**:219-269.

Goodall, MC (1960) Performance of a stochastic net. *Nature* **185**:557-558.

Grossberg, S (1982) Processing of expected and unexpected events during conditioning and attention: A psychophysiological theory. *Psychological Review* **89**:529-572.

Hopfield, JJ (1982) Neural networks and systems with emergent selective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America* **79**:2554-2558.

Hopfield, JJ (1984) Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences of the United States of America* **81**:3088-3092.

Krauth, W & Mézard, M (1987) Learning algorithms with optimal stability for neural networks. *Journal of Physics* **A20**:L745-L752.

Leen, TK (1991) Dynamics of learning in recurrent feature-discovery networks. In RP Limann, JE Moody & DS Touretzky, editors, *Advances in Neural Information Processing Systems, 3*. San Mateo, CA:Morgan Kaufmann.

Levy, WB (1996) A sequence predicting CA3 is a flexible associator that learns and uses context to solve hippocampal-like tasks. *Hippocampus* **6**:579-590.

Levy, WB & Steward, D (1983) Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus. *Neuroscience* **8**:791-797.

Linsker, R (1988) Self-organization in a perceptual network. *Computer* **21**:105-117.

MacKay, DJC & Miller, KD (1990) Analysis of Linsker's application of Hebbian rules to linear networks. *Network: Computation in Neural Systems* **1**:257-299.

Markram H, Lubke J, Frotscher M, Sakmann B (1997) Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. Science 275:213-215.

Mehta, MR, Barnes, CA & McNaughton, BL (1997) Experience-dependent, asymmetric expansion of hippocampal place fields. *Proceedings of the National Academy of Science USA* **94**:8918-8921.

Miller, KD (1994) A model for the development of simple cell receptive fields and the ordered arrangement of orientation columns through activity-dependent competition between on- and off-center inputs. *Journal of Neuroscience* **14**:409-441.

Miller, KD, Keller, JB & Stryker, MP (1989) Ocular dominance column development: Analysis and simulation. *Science* **245**:605-615.

Montague, PR & Sejnowski, TJ (1994) The predictive brain: Temporal coincidence and temporal order in synaptic learning mechanisms. *Learning and Memory* **1**:1-33.

Nicholls, JG, Martin, R & Wallace, BG (1992) *From Neuron to Brain: A Cellular and Molecular Approach to the Function of the Nervous System*. Sunderland, MA:Sinauer Associates, P.611.

Oja, E (1982) A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology* **16**:267-273

Oja, E (1989) Neural networks, principal components and subspaces. *International Journal of Neural Systems* **21**:61-68.

Oja, E (1992) Principal components, minor components and linear neural networks. *Neural Networks* **5**:927-935.

Oja, E & Karjunen, J (1985) On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of Mathematical Analysis and Applications* **106**:69-84.

Plumbley, MD (1993) Efficient information transfer and anti-Hebbian neural networks. *Neural Networks* **6**:823-833.

Poggio, T (1990) A theory of how the brain might work. *Cold Spring Harbor Symposium on Quantitative Biology* **55**:899-910.

Poggio, T & Girosi, F (1990) Networks for approximation and learning. *Proceedings of the IEEE* **78**:1481-1497.

Poggio, T & Girosi, F (1990) Regularization algorithms for learning that are equivalent to multilayer networks. *Science* **247**:978-982.

Rosenblatt, F (1958) The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* **65**:386-408.

Sanger, TD (1989) Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks* **2**:459-473.

Sejnowski, TJ (1977) Storing covariance with nonlinearly interacting neurons. *Journal of Mathematical Biology* **4**:303-321.

Turrigiano, G (1998) Homeostatic plasticity in neuronal networks: The more things change, the more they stay the same. *Trends in Neuroscience* (in press).

Wallis, G & Baddeley, P (1997) Optimal, unsupervised learning in invariant object recognition. *Neural Computation* **9**:883-894.

Wallis, G, Rolls, E & Foldiak, P (1993) Learning invariant responses to the natural transformations of objects. *Proceedings of the International Joint Conference on Neural Networks* **2**:1087-1090.

Williams, RJ (1985) *Feature Discovery through Error-Correction Learning*. Technical Report 8501, Institute of Cognitive Science, University of California, San Diego.

Zhang, LI, Tao, HW, Holt, CE, Harris, WA & Poo M-m (1998) A critical window for cooperation and competition among developing retinotectal synapses. *Nature* **395**:37-44.

8.5 Exercises

1. Simulate the course of Hebbian learning for the case of figure 8.2. For initial weights expressed as $(w + \zeta, w - \zeta)$, what range of values of w and ζ lead to saturation at $(1, 1)$. Can you predict this theoretically? If the off-diagonal term in the correlation matrix is -2 instead of -0.4 and there are no saturation boundaries, what happens to the sum of the weights? Could this be used as a way of normalizing the weights?

2. Show that the averaged form of the single-trial Oja rule in equation 8.15 is given by the expression in equation 8.16. Prove that if it converges, the averaged learning rule produces a set of weights proportional to an eigenvector of the correlation matrix $\mathbf{Q}$ and normalized so that $|\mathbf{w}| = 1$.

3. Simulate the simple model of ocular dominance in figure 8.7, including a cortical interaction described by the difference-of-Gaussians function described in the caption to figure 8.4. How does the resulting pattern compare with that generated by taking binary noise and filtering it through a spatial filter composed by the same difference-of-Gaussians function.

4. Construct two-dimensional input data sets like those shown in figure 8.10 and use them to train a two-input, one output linear network using correlation- and covariance-based Hebbian learning rules with

multiplicative normalization. Compare the final outcome for the weights with the principal components of the data when the mean of the input distribution is zero and when it is nonzero.

5. Repeat exercise 4 for a data set with zero mean, but with subtractive normalization. When does this algorithm produce a weight vector aligned with the principal component axis of the input data set, and when does it fail to do so. Why does the weight vector sometimes fail to align with the principal component axis?

6. Show that if the estimate of equation 8.37 is written as $\mathbf{u}_{\text{approx}} = \sum(\mathbf{u} \cdot \mathbf{g}_\mu)\mathbf{g}'_\mu$, the averaged squared error in equation 8.38 is minimized when $\mathbf{g}_\mu = \mathbf{g}'_\mu$ for all μ.

7. Explore the effect of the learning rate ϵ on the gradient descent minimization in figure 8.11B. What happens if $\epsilon > 2$, and why?

8. Train the feedforward network of figure 8.11A to produce the output $v = \cos(0.6s)$ when the input tuning curves are given as in the caption to figure 8.12. Train the network by using the stochastic delta learning rule with s values chosen randomly in the range between -10 and 10.

9. Construct a perceptron (equation 8.59) that classifies 10 binary inputs according to whether their sum $\sum u_a$ is positive or negative. Use a random set of binary inputs during training and compare the performance (both the learning rate and the final accuracy) of the Hebbian, delta, and perceptron learning rules. Repeat this training protocol, but this time attempt to make the output of the perceptron classify according to the parity of the inputs, which is the sign of their product $\prod u_a$. Why is this example so much harder than the first case?

10. Build a Hopfield associative memory network by iterating the rule 8.69 for 100 interconnected units. Construct P memory states by randomly assigning plus and minus values to the N elements of each $\mathbf{v}_m$. Using these memory vectors, set the matrix of synaptic weights according to equation 8.71 (with diagonal elements set to zero). To measure how close the state of the network at time t, $\mathbf{v}(t)$, is to a particular memory state, define the overlap function $q(t) = \mathbf{v}(t) \cdot \mathbf{v}_m / N$. This is equal to 1 if $\mathbf{v}(t) = \mathbf{v}_m$, near zero if $\mathbf{v}(t)$ is unrelated to $\mathbf{v}_m$ and equal to -1 if $\mathbf{v}(t) = -\mathbf{v}_m$. Create an initial state $\mathbf{v}(0)$ with an overlap with memory state $\mathbf{v}_1$, $q(0)$, near one. As the network evolves according to equation 8.69, plot $q(t)$. Final values of $q(t)$ near one indicate successful recovery of the memory state. Determine how close $q(0)$ must be to one in order to achieve successful memory recovery for different values of P. Start with $P = 1$ and increase it until memory recovery fails even for $q_0 = 1$. At what P value does this occur?