

Exercises

Chapter 8

1. Simulate the course of Hebbian learning for the case of figure 8.3. Find the ranges of initial weight values, (w_1, w_2) , that lead to saturation at $(1, 1)$. Can you predict the result analytically? 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 8.16 is given by

$$\tau_w \frac{d\mathbf{w}}{dt} = \mathbf{Q} \cdot \mathbf{w} - \alpha(\mathbf{w} \cdot \mathbf{Q} \cdot \mathbf{w})\mathbf{w}.$$

Prove that if it converges, the averaged learning rule produces a set of weights proportional to an eigenvector of the correlation matrix \mathbf{Q} , normalized so that $|\mathbf{w}|^2 = 1/\alpha$.

3. Simulate the ocular dominance model of figure 8.7 using a subtractively normalized version of equation 8.31 (i.e. equation 8.14) with saturation limits at 0 and 1, and cortical interactions generated as in figure 8.8 from

$$\mathbf{K}_{aa'} = \exp\left(-\frac{(a - a')^2}{2\sigma^2}\right) - \frac{1}{9} \exp\left(-\frac{(a - a')^2}{18\sigma^2}\right),$$

where $\sigma = 0.066$ mm. Use 512 cortical cells with locations a spread evenly over a nominal 10 mm of cortex, and periodic boundary conditions (this means that you can use Fourier transforms to calculate the effect of the cortical interactions). Also use the discrete form of equation 8.31

$$\mathbf{W} \rightarrow \mathbf{W} + \epsilon \mathbf{K} \cdot \mathbf{W} \cdot \mathbf{Q}$$

with a learning rate of $\epsilon = 0.01$. Plot \mathbf{w}_- as it evolves from near $\mathbf{0}$ to the final form of ocular dominance. Calculate the magnitude of the discrete Fourier transform of \mathbf{w}_- . Repeat this around 100 times, work out the average of the magnitudes of the Fourier transforms, and compare this to the Fourier transform of \mathbf{K} .

4. Construct two-dimensional input data sets similar to those shown in figure 8.4 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 and saturation. Start with initial values for the weights that are chosen randomly over the full range from 0 to their saturation

limit. 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. Consider minimizing the function $E(w) = (w - 2)^2$ using the gradient descent rule for w ,

$$w \rightarrow w - \epsilon \frac{dE}{dw} .$$

Plot $E(w)$ together with the trajectories of w starting from $w = 5$ for $\epsilon = 0.01, 0.1, 1, 2, 3$. Why does learning diverge as ϵ gets large?

7. Consider $E(\mathbf{w}) \propto \langle (h(s) - \mathbf{w} \cdot \mathbf{f}(s))^2 \rangle$, as in equation 8.52, in the case that matrix $\langle \mathbf{f}(s)\mathbf{f}(s) \rangle$ is invertible. An extended delta rule can be written as

$$\mathbf{w} \rightarrow \mathbf{w} + \langle (h(s) - \mathbf{w} \cdot \mathbf{f}(s)) \mathbf{H} \cdot \mathbf{f}(s) \rangle ,$$

where \mathbf{H} is a matrix that generalizes the learning rate ϵ of the standard delta rule. For what matrix \mathbf{H} does this rule go from any initial value \mathbf{w} to the optimal weights in one single step. This amounts to a form of the Newton-Raphson method.

8. Train the feedforward network of figure 8.13 to produce the output $v = \cos(0.6s)$ when the input tuning curves are given as in the caption to figure 8.14. Train the network by using the stochastic delta learning rule (equation 8.61) with s values chosen randomly in the range between -10 and 10.
9. Construct a perceptron (equation 8.46) 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 (equation 8.47), 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?