

Homework #1

Instructions: Answer all questions below. You do not need to turn in your Matlab code, but your answers must include enough detail so that I know that you performed the computer simulations, you obtained the correct results, and you know why you obtained these results. Please error on the side of providing too much documentation as opposed to too little.

Question 1

Consider the following set of three training patterns:

Pattern	Input	Output
-----	-----	-----
1	-x +x +x -x -x +x +x -x	1 0 0
2	-x -x +x +x -x -x +x +x	0 1 0
3	-x +x -x +x -x +x -x +x	0 0 1

where $x = 0.354$ (that is, $-x = -0.354$ and $+x = 0.354$). Train a linear network with 8 input units and 3 output units on this data using the supervised Hebb rule. Initialize the network by setting the weights to zero. Do not use bias weights. Use a learning rate of 1.0 (that is, in the Hebb rule let $\epsilon = 1.0$). Present the training patterns to the network in a random order. First train the network by going through the data set exactly once (that is, do one epoch of training). At the end of this epoch, test the network's performance (that is, compare the network's outputs with the desired outputs). Is the performance good or poor? Why? Now run nine additional epochs of training. What happens? Does the network's performance improve, degrade, or stay the same? Do the network's weights change significantly from their values after one epoch of training to their values after ten epochs of training?

The three training patterns may be considered to be prototypes for three categories. By randomly perturbing the input portions of each prototype, we can create a large set of training patterns. Here is such a set:

Pattern	Input	Output
-----	-----	-----
1	+x +x +x -x -x +x +x -x	1 0 0
2	-x +x -x -x -x +x +x -x	1 0 0
3	-x +x +x +x -x +x +x -x	1 0 0
4	-x +x +x -x -x -x +x -x	1 0 0
5	-x +x +x -x -x +x -x -x	1 0 0
6	-x +x +x +x -x -x +x +x	0 1 0

7	-x -x +x -x -x -x +x +x	0 1 0
8	-x -x +x +x +x -x +x +x	0 1 0
9	-x -x +x +x -x +x +x +x	0 1 0
10	-x -x +x +x -x -x -x +x	0 1 0
11	+x +x -x +x -x +x -x +x	0 0 1
12	-x +x +x +x -x +x -x +x	0 0 1
13	-x +x -x -x -x +x -x +x	0 0 1
14	-x +x -x +x -x -x -x +x	0 0 1
15	-x +x -x +x -x +x -x -x	0 0 1

As above, train a linear network with 8 input units and 3 output units on this data using the supervised Hebb rule; initialize the network by setting the weights to zero; do not use bias weights; present the patterns in a random order. In this simulation, use a learning rate of 0.2 and only train the network for a single epoch. Does the network's performance differ on this training set than when it was trained using just the prototypes? If so, how and why?

The network trained with the larger training set has never seen the prototypical patterns. Without further training (that is, don't change the weights), evaluate the network's performance on the three prototypes. Does it perform well? Why or why not?

Are the input portions of the prototypes orthogonal, linearly independent, and/or linearly dependent? Prove your answer. Does this influence the network's performance? If so, how? Are the output portions of the prototypes orthogonal, linearly independent, and/or linearly dependent? Prove your answer. Does this influence the network's performance? If so, how?

Now redo the above simulations with some new variables. First, does the network's performance change when the learning rate is 1/10, 1/2, 1, or 2? If so, how? Second, does it matter if you train the network on the training patterns in the given order versus when the patterns are presented in a random order? Third, does the network's performance change when the output units are each provided with a bias weight?

Question 2

Consider the following set of training patterns:

Pattern	Input	Output
-----	-----	-----
1	+x -x +x -x +x +x -x -x	+1 +1 -1 -1 +1 +1 -1 -1
2	+x +x -x -x +x -x +x -x	+1 +1 +1 +1 -1 -1 -1 -1
3	+x -x +x -x +x +x +x +x	+1 -1 -1 +1 +1 -1 -1 +1

Train a linear network with 8 input units and 8 output units on this data using the supervised Hebb rule. As above, initialize the network by setting the weights to zero; do not use bias weights; use a learning rate of 1.0; present the training patterns in a random order. After one epoch of training, how is the network's performance? After ten epochs of training, how is its performance? Overall, how does this network's performance compare with the performance of the network trained with

the prototypes from Question 1? If they are roughly the same, why? If one is better than the other, why?

Question 3

Consider the training pattern:

Input	Output
-----	-----
0.6 0.8	3.0

The weights of a linear network with two input units and one output unit form a vector $\mathbf{w} = [w_1 \ w_2]^T$ where w_i is the weight on the connections from the i^{th} input unit to the output unit. Note that the three weight vectors $[1.8 \ 2.4]^T$, $[0.9 \ 3.075]^T$, and $[2.7 \ 1.725]^T$ all enable the network to correctly perform the training pattern (please prove this to yourself). In fact, there are an infinite number of weight vectors that enable the network to correctly perform the pattern. What geometric property do all of these vectors have in common?

Question 4

Consider the following two-dimensional data:

x1	x2
--	--
-0.90000	-0.76984
-0.70000	-0.89574
-0.50000	-0.56198
-0.30000	-0.26177
-0.10000	0.03305
0.10000	0.02737
0.30000	0.18939
0.50000	0.63889
0.70000	0.72811
0.90000	0.89430

Train a linear network with 2 input units and 1 output unit on this data using the unsupervised Hebb rule due to Oja:

$$w_i(t+1) = w_i(t) + \epsilon[y(t)x_i(t) - y(t)^2w_i(t)]. \quad (1)$$

Initialize the network by setting the weights to random values with the constraint that the length of the weight vector $\mathbf{w} = [w_1 \ w_2]^T$ equals one. Do not use bias weights. Present the training patterns to the network in a random order. Try different learning rates, and try different numbers of epochs. Does the weight vector \mathbf{w} converge to the eigenvector of the data covariance matrix with the largest eigenvalue?

Question 5

Consider the following two-dimensional data:

x1	x2
--	--
0.73885	1.34754
1.47684	0.29641
-2.40937	-1.92435
0.03902	0.56988
-0.96952	2.14645
0.56623	0.29912
0.33784	0.33839
-1.29562	1.20109
-0.01690	0.08101
1.15710	-1.40003

Plot these ten points on a two-dimensional graph. Next, perform principal component analysis on this dataset (hint: you will want to use the Matlab functions “cov” and “eigs”), and determine the “optimal” one-dimensional representation of each point (i.e., project each point onto the eigenvector of the data’s covariance matrix with the largest eigenvalue). Plot these ten points on a one-dimensional graph. Would it be useful to use this data to train a linear network with two input units and one output unit via the unsupervised Hebb rule? Why or why not?

Question 6

Consider the following two-dimensional data:

x1	x2
--	--
-0.90000	-0.76984
-0.70000	-0.89574
-0.50000	-0.56198
-0.30000	-0.26177
-0.10000	0.03305
0.10000	0.02737
0.30000	0.18939
0.50000	0.63889
0.70000	0.72811
0.90000	0.89430

Plot these ten points on a two-dimensional graph. Next, perform principal component analysis on this dataset (hint: you will want to use the Matlab functions “cov” and “eigs”), and determine the “optimal” one-dimensional representation of each point (i.e., project each point onto the eigenvector of the data’s covariance matrix with the largest eigenvalue). Plot these ten points on a one-dimensional graph. Would it be useful to use this data to train a linear network with two input units and one output unit via the unsupervised Hebb rule? Why or why not?