

Networks that Learn

Total points 20

1. (Question Variation 2: Numbers different in different variations) Suppose we have a linear feedforward network with two input nodes and one output node. Let's say that we are learning our weight vector \mathbf{w} and that we are using the Hebb rule. 3 points

Suppose the input correlation matrix Q is:

$$Q = \begin{bmatrix} 0.15 & 0.1 \\ 0.1 & 0.12 \end{bmatrix}$$

If we allow learning to go on for a long period of time, which of these could be a final weight vector \mathbf{w} ?

(Hint: See lecture 7.1 and use Matlab's "eig" command)

- ☒ $\mathbf{w} = \begin{bmatrix} -1.5155 \\ -1.3051 \end{bmatrix}$
- ☐ $\mathbf{w} = \begin{bmatrix} 0.8944 \\ 1.7889 \end{bmatrix}$
- ☐ $\mathbf{w} = \begin{bmatrix} -1.5764 \\ -1.2308 \end{bmatrix}$
- ☐ $\mathbf{w} = \begin{bmatrix} 1.0515 \\ 1.7013 \end{bmatrix}$

2. What does the weight vector we found in Question 1 tell us? 1 point

- ☒ None of these options
- ☐ That the mean of the input distributions is (0,0)
- ☐ That the time constant τ_w was 2
- ☐ That this Hebbian learning system is engaged in long-term depression of the outputs

3. The "learning rate" as mentioned in lecture 7.2 is used in many different contexts to denote the sensitivity of a parameter estimate to new data during online learning. Check **all** of the following which are true: 2 points

- ☒ A higher learning rate makes the parameter estimate more influenced by noise during online estimation.
- ☒ A constant learning rate allows the parameter estimate to remain sensitive to new data.
- ☐ A higher learning rate makes the parameter estimate less influenced by noise during online estimation.
- ☐ The learning rate does not affect the parameter estimate's sensitivity to noise.

4. In the lectures, we saw multiple algorithms which use a two-step process for parameter estimation. EM is one such algorithm, consisting of the E and M steps. In each step, we iteratively update our estimates of parameters. Why do we need to alternate between the two steps? What justifies this approach? 1 point

(Hint: You may want to search "EM algorithm" on the web)

- ☒ We need the alternating steps because the two sets of parameters are mutually dependent on each other, necessitating that they be optimized together. This is justified by the fact that these algorithms will always attempt to move towards a better solution with each iteration.
- ☐ We don't actually need the alternating steps - we could get away with optimizing one set of values at a time, and doing it only once; we alternate because it is simple to program the alternation as a loop. This is justified by the fact that these algorithms will always arrive at the globally optimal solution.
- ☐ We need the alternating steps because the two sets of parameters are mutually dependent on each other, necessitating that they be optimized together. This is justified by the fact that these algorithms will always arrive at the globally optimal solution.
- ☐ We don't actually need the alternating steps - we could get away with optimizing one set of values at a time, and doing it only once; we alternate because it is simple to program the alternation as a loop. This is justified by the fact that these algorithms will always attempt to move towards a better solution with each iteration.

5. In the next five questions, we'll implement Oja's Hebb rule for a single neuron and explore some of its properties. 1 point

Recall that Oja's rule is:

$$\tau_w \frac{d\mathbf{w}}{dt} = v\mathbf{u} - \alpha v^2 \mathbf{w},$$

where $v = \mathbf{u} \cdot \mathbf{w}$.

We will implement the discrete time version of Oja's rule in Matlab or Octave. To do this, first rewrite Oja's rule using discrete time rather than continuous time. Which of the following equations represents the discrete time version of Oja's rule? (Let $\eta = \frac{1}{\tau_w}$).

- ☒ $\Delta \mathbf{w} = \Delta t \eta (v\mathbf{u} - \alpha v^2 \mathbf{w})$
- ☐ $\Delta \mathbf{w} = \Delta t (v\mathbf{u} - \alpha v^2 \mathbf{w})$
- ☐ $\frac{d\mathbf{w}}{dt} = \eta (v\mathbf{u} - \alpha v^2 \mathbf{w})$
- ☐ $\Delta \mathbf{w} = \eta (v\mathbf{u} - \alpha v^2 \mathbf{w})$


6. Continued from Question 5. 1 point

Now, in order to use Oja's rule, we need to translate the discrete time version into an update rule. Which of the following equations represents the discrete update equation for Oja's rule?

- ☒ $\mathbf{w}_{i+1} = \mathbf{w}_i + \Delta t \eta (v\mathbf{u} - \alpha v^2 \mathbf{w})$
- ☐ $\mathbf{w}_{i+1} = \mathbf{w}_i + \Delta t (v\mathbf{u} - \alpha v^2 \mathbf{w})$
- ☐ $\mathbf{w}_{i+1} = \mathbf{w}_i + \eta (v\mathbf{u} - \alpha v^2 \mathbf{w})$
- ☐ $\mathbf{w}_{i+1} = \mathbf{w}_i + v\mathbf{u} - \alpha v^2 \mathbf{w}$

7. Continued from Question 5. 5 points

In this question, we will use the update rule we just derived to implement a neuron that will learn from two dimensional data that is given in the following Matlab file:

**c10p1**
MAT File

(This file is provided as part of the [exercises](#) from the Dayan and Abbott textbook recommended for the course).

c10p1.mat contains 100 (x, y) data points.

Move c10p1.mat to your Matlab or Octave directory and use the following command to load the data (note that you must include the '-ascii' option for the file to load correctly):


```
1 load('-ascii', 'c10p1.mat')
```

You may plot the data points contained in c10p1.mat using the following command:


```
1 scatter(c10p1(:,1), c10p1(:,2))
```

The equivalent python pickle files are:

Python 2.7:

**c10p1**
PICKLE File

Python 3.4:

**c10p1**
PICKLE File

and can be loaded in the usual way:

```
1 import pickle
2 with open('c10p1.pickle', 'rb') as f:
3     data = pickle.load(f)
```

(NOTE ON DOWNLOADING CODE AND DATA: Currently, downloaded files are automatically renamed to begin with a long string of random characters (we hope to have this fixed soon). Sometimes the file type is also changed. In order to ensure that all of the files in the quizzes work correctly, make sure that after downloading each file you rename it to the file name shown in the original quiz question. If you still have trouble getting any of the files to open feel free to search or inquire on the class Discussion Forums.)

Assume our neuron receives as input the two dimensional data provided in c10p1.mat, but with the mean of the data subtracted from each data point (the mean of all x values should be subtracted from every x value and the mean of all y values should be subtracted from every y value). You should perform this zero-mean centering step and then display the points again to verify that the data cloud is now centered around $(0, 0)$.

Implement the update rule derived in the previous question in Matlab or Octave. Let $\eta = 1, \alpha = 1$, and $\Delta t = 0.01$. Start with a random vector as \mathbf{w}_0 . In each update iteration, feed in a data point $\mathbf{u} = (x, y)$ from c10p1. If you've reached the last data point in c10p1, go back to the first one and repeat.

Typically, you would keep updating \mathbf{w} until the change in \mathbf{w} , given by $\text{norm}(\mathbf{w}(t+1) - \mathbf{w}(t))$, is negligible (i.e., below an arbitrary small positive threshold), indicating that \mathbf{w} has converged. However, since you are implementing this as an online learning algorithm, you may prematurely detect convergence using this method. Instead, you may just run the algorithm for 100,000 iterations.

Run your code multiple times. Which of the following describes the behavior of \mathbf{w} and why does this happen?

Hint: Consider the eigenvectors of the correlation matrix of the mean-centered data. (The correlation matrix of a data matrix \mathbf{X} , where rows indicate separate samples, is $\mathbf{X}^T \mathbf{X} / N$, where N is the number of samples. You can calculate its eigenvalues using `eig()`.) If the data is mean-centered, the correlation matrix will be the same as the covariance matrix.

- ☒ The correlation matrix has only one principal eigenvector, but there are two vectors of length $\frac{1}{\sqrt{\alpha}}$ that are parallel to this eigenvector. \mathbf{w} can converge to either of these two vectors.
- ☐ The input data is two dimensional and therefore the correlation matrix has two eigenvectors. \mathbf{w} can converge to either of these eigenvectors regardless of their corresponding eigenvalues.
- ☐ \mathbf{w} always converges to the same unique vector because the correlation matrix of the input data has only one principal eigenvector.
- ☐ There is only one principal eigenvector of the correlation matrix, but there are two vectors of length $\frac{1}{\sqrt{\alpha}}$ that are perpendicular to this eigenvector. \mathbf{w} converges to either of these two vectors.

8. Continued from Question 5. 3 points

What happens when the data is not zero-mean centered before the learning process?

In order to more fully explore the behavior of the Oja's rule when the data isn't mean centered, you should adjust the mean of the data a few times and observe the behavior of the learning rule. You can adjust the mean of the data by adding a constant to every x component of the data and a different constant to every y component of the data.

- ☒ The two vectors that \mathbf{w} converges to in different runs of the algorithm are parallel to the vector that points roughly towards the mean of the data.
- ☐ The two vectors that \mathbf{w} converges to in different runs of the algorithm have the same direction as those found when the data is mean centered.
- ☐ The two vectors that \mathbf{w} converges to in different runs of the algorithm are exactly the same as those found when the data is mean centered.
- ☐ \mathbf{w} now converges to the same vector every time the code is run.

9. Continued from Question 5. 3 points

What happens when the pure Hebb rule is used instead of Oja's rule? You can explore what happens by removing the subtractive term $-\alpha v^2 \mathbf{w}$ in your code and running the code.

- ☒ The vectors found by the learning rule have the same direction as those found by Oja's rule, but the length grows without bound as a function of the number of iterations.
- ☐ The vectors found by the learning rule are identical to those found by the original learning rule.
- ☐ The vectors found by the learning rule are completely different from those found by the original learning rule: both the direction and length are different.
- ☐ \mathbf{w} converges to the same vector every time the algorithm is run instead of converging to two different vectors whenever the algorithm is run.

Coursera Honor Code [Learn more](#)

☒ I understand that submitting work that isn't my own may result in permanent failure of this course or deactivation of my Coursera account.

Daniel Engbert

Use the name on your government issued ID

Submit

Save draft