1 Theory Problem - Perceptron

a) Give a proof of the Perceptron Convergence Theorem anyway you can.

Let $z(n)$ be the $x(n)$ space transformed to include the bias. Recall from the text and lecture that the perceptron learning rule is given by:

$$\hat{w}(n+1) = \hat{w}(n) + \eta \delta(n) z(n)$$

where $\delta(n)$ is 1 if the $n^{th}$ example is misclassified by the current weight vector and 0 otherwise.

Now let $\hat{z}(n)$ be $z(n)$ normalized, namely:

$$\hat{z}(n) = \frac{z(n)}{||z(n)||}$$

Let $w_+$ be a solution vector such that $w_+^T \hat{z}(n) \geq 1$ for all $n$. Now let us suppose that $\hat{z}(n)$ is misclassified. Consider the distance between the new weight vector and the solution vector. Set $\eta = 1$.

$$||\hat{w}(n+1) - w_+||^2 = ||(\hat{w}(n) + \hat{z}(n)) - w_+||^2$$

$$= ||(\hat{w}(n) - w_+ + \hat{z}(n)||^2$$

$$= ||\hat{w}(n) - w_+||^2 + 2\hat{z}(n)^T(\hat{w}(n) - w_+) + 1$$

$$= ||\hat{w}(n) - w_+||^2 + 2\hat{z}(n)^T\hat{w}(n) - 2\hat{z}(n)^Tw_+ + 1$$

Equation (4) shows that the distance between our new weight vector and the solution vector decreases by at least one on every iteration. This means that the algorithm will
converge in at most $||\hat{w}_0 - w_+||^2$ steps. Setting $\hat{w}_0 = 0$ the algorithm will converge in at most $||w_+||^2$ steps.

Thus I have shown through algebraic manipulation that the perceptron will always converge in finite time if there is a solution for $\eta = 1$.

b What effect will varying the learning parameter have on convergence? Why?

If I had originally defined $w_+$ such that the inner product of it and every example to be $\geq b$ and had carried the $\eta$ through the manipulations you would see that with a value of $\eta < 2b$ the algorithm converges to a solution in time $\frac{||w_+||^2}{\eta(2b-\eta)}$ which is minimized for $\eta = b$. By observation varying $\eta$ changes the speed of convergence. It seemed that larger $\eta$ resulted in faster convergence, but not all the time.

2 Computer Problem - Perceptron Experimentation

a Using you’re your own code and data structures, implement a simulation of the Perceptron Learning Rule on a single neuron with two data inputs and one bias input. See attached code listing “main.c”.

b Plot the included training data as an x-y scatter plot and determine by visual inspection if they are linearly separable.

By examining figure (1) it is obviously linearly separable.

c Train the Perceptron with the included training data. Produce plots of the three weights and the number of errors as a function of simulator epoch. What effect does randomizing the order of the training data have on the weight convergence.

By examining figures (2 - 5) we see that randomization can minorly affect the time to convergence. There does not appear to be a clear trend towards better performance as far as I can see. As with most randomized approaches, we would expect the average behavior to be fairly nice if we present randomized orderings of data. Randomization can often help avoid a malicious ordering of data points.

d Run this multiple times with different presentation orders and initial weights, plotting the range of variation of error count as a function of epochs.

3 Computer Problem - Delta-Rule Experimentation

a Similar to above, using you Rre your own code and data structures again, implement a simulation of the Delta-Rule Learning Rule on a single neuron with two data inputs and one bias input. See attached code listing “ass3.m”.

b Plot the included training data as an x-y scatter plot and determine by visual inspection if they are linearly separable.

By examining figure (1) it is obviously linearly separable.

c Train the Perceptron with the included training data. Produce plots of the three weights and the number of errors as a function of simulator epoch. What effect does randomizing the order of the training data have on the weight convergence.

By examining figures (2 - 5) we see that randomization can minorly affect the time to convergence. There does not appear to be a clear trend towards better performance as far as I can see. As with most randomized approaches, we would expect the average behavior to be fairly nice if we present randomized orderings of data. Randomization can often help avoid a malicious ordering of data points.

d Run this multiple times with different presentation orders and initial weights, plotting the range of variation of error count as a function of epochs.
b Train the neuron with the included training data. Produce plots of the three weights and the total RMS error as a function of simulator epoch. What effect does randomizing the order of the training data have on the weight convergence.

Again, I see no definative trend towards consistent better performance, but would expect that the average performance would be best if data was presented in a random order.

d Run this multiple times with different presentation orders and initial weights, plotting the range of variation of total RMS error value as a function of epochs.
Figure 2: Plot of weights and errors as a function of simulator epoch for a unique presentation of data

Figure 3: Plot of weights and errors as a function of simulator epoch for a unique presentation of data
Figure 4: Plot of weights and errors as a function of simulator epoch for a unique presentation of data

Figure 5: Plot of weights and errors as a function of simulator epoch for a unique presentation of data
Figure 6: Plot of weights and errors as a function of simulator epoch for a presentation of data including one misclassified element

Figure 7: Plot of errors as a function of simulator epoch for multiple unique presentations and weights
Figure 8: Plot of weights and RMS error as a function of simulator epoch for a unique presentation of data.

Figure 9: Plot of weights and RMS error as a function of simulator epoch for a presentation of data including a misclassified element.
Figure 10: Plot of RMS error as a function of simulator epoch for a unique presentation of data and unique weights