Question 3

Given:

- Pattern Classification
- Delta Rule
- Input layer has 2 neurons
- Output Layer has 1 neuron
- Linearly decreasing learning rate alpha
    - .9 to 10^-5 over the training set
- 100 epochs

Find:

- Network Design
- Final Weights
- Plot MSE vs Epoch
- Find Error rate
- Plot test data and classification boundaries

Iteration 1:

Net Construction:

Because our input values are in a different form from our target values (2 inputs to 1 output), we will be using a hetero-associative net for pattern classification. Our inputs will be 2 bits. Our outputs will have one bit.

Weights will be updated using the delta rule. The weight matrix will be of size 2x1. The stopping condition for training will be 100 epochs. Training and testing data have been given in the form of 2 x 1500 and 2 x 2500 matrices. Decimal values have been given for these input sets.

We will define one epoch as a full run through the entire training set, which would be 1500 individual vectors. By defining one individual vector as one iteration through the phase, our entire phase will consist of 150000 iterations.

During the training phase, a plot will be made to show the progression of the MSE over the total number of epochs. During the testing phase, we will calculate outputs based on our final weights and determine if they match the target values. A plot will be made to show the positions of the testing set and the final decision boundary created with the final weights.

Activation Function:

This iteration will be using the bipolar activation function. It is very simple, outputting 1 if the input is greater than one and -1 for any other cases.

```
function [output] = bipolar_act(yin)

    if yin > 0
        output = 1;
    else
        output = -1;
    end
end
```

Algorithm:

First, we will begin by initializing the data into test and training sets.

```
S = Training_Data;
t = Target_Data;
STest = Test_Data;
tTest = Test_Target_Data;
```

Next, we will define the epochs as a vector from 1 to 100 and allocate alpha in the same way. Alpha will be decreased over the entire training phase, which equals to 150000 iterations. I chose this option as opposed to decreasing alpha over each epoch because distributing alpha over the entire phase seemed to give more consistent results in terms of MSE. The implementation where alpha decreases over the course of the epoch is also shown below. The differences in MSE can be seen by viewing the graphs 'MSE vs Epochs for Alpha over Epoch' and 'MSE vs Epochs for Alpha over Phase' in the Plots folder. The MSE in the phase graph initially decreases sharply, then slowly decreases at a steady rate. The MSE in the epoch graph is chaotic over time.

Over training phase:

```
epochs = 1:100;
alphastep = (.9 - 1e-5)/(150000-1);
alpha = .9:-alphastep:1e-5;
```

Over an epoch:

```
epochs = 1:100;
alphastep = (.9 - 1e-5)/(1500-1);
alpha = .9:-alphastep:1e-5;
```

Next, we will initialize the weights to zero. We will also define variables to hold the total squared error, MSE, and total iterations.

```
W = zeros(2,1);
totalerrsq = 0;
MSE = zeros(1,100);
iter = 1;
```

Next, we will train on the training set for 100 epochs. The training algorithm will first choose the alpha for this iteration. Then, it will calculate the yin value and apply the bipolar activation function to it. Next, the algorithm will calculate the error squared between the input and target values and add it to the total error squared. Finally, it will update the weights with the delta rule and update the iteration count.

```
for x = epochs
    for n = 1:length(S(1,:))
        a = alpha(iter);
        yin = W.' * S(:,n);
        yin = bipolar_act(yin);
        err = t(n) - yin;
        errsq = err^2;
        totalerrsq = totalerrsq + errsq;
        W = W + a * err * S(:,n);
        iter = iter + 1;
    end
    MSE(x) = totalerrsq/(x*n);
end
```

Next, the net will calculate the output values for each of the training sets and see if they match up with the target values. We will first pre allocate a matrix to hold the Booleans showing whether a match occurred at test vector xi. Then, we will calculate the output values and turn on each value in the match matrix for every yout that matches with the target values.

```
for n = 1:length(STest(1,:))
    yout = W.' * STest(:,n);
    yout = bipolar_act(yout);
    if yout == tTest(n)
        match(n) = 1;
    end
end
```

We will then calculate the accuracy by summing all the values in the match matrix, then dividing it by the length of the test target values. We will subtract the accuracy from 1 to find the error rate. Next, we will define the decision boundary using an anonymous function. Finally, we will plot the MSE versus Epochs on one figure. On the other figure, we will plot the testing input values and the final decision boundary.

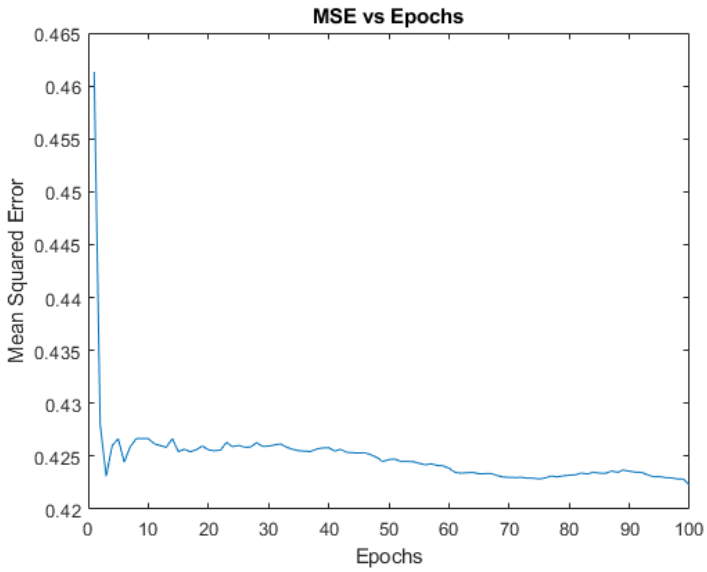
```
errorRate = 1 - sum(match)/length(STest(1,:));

decision = @(x) -(W(1)/W(2)).* x;

figure(1)
    plot(MSE)
        xlabel('Epochs')
        ylabel('Mean Squared Error')
        title('MSE vs Epochs')

figure(2)
hold on
    scatter(STest(1,:),STest(2,:))
    fplot(decision)
        xlabel('X1')
        ylabel('X2')
        title('Test Points with Decision Bounduary')
hold off
```

Final Weights:

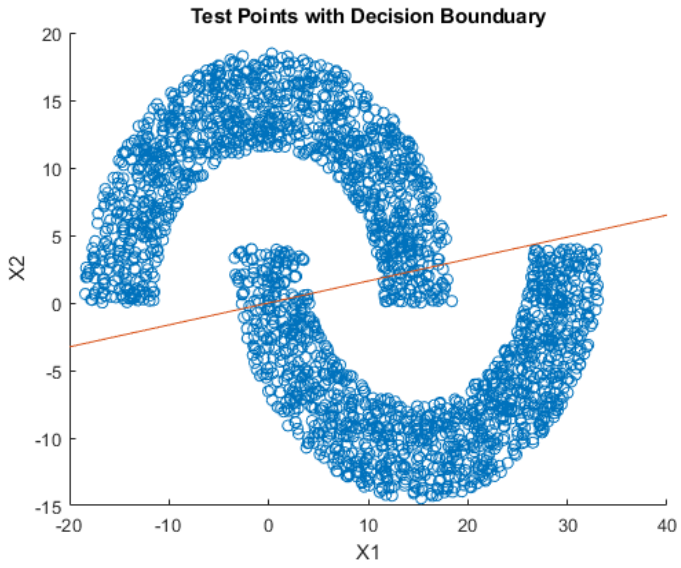
W = [- 0.0522;

0.3211];

MSE vs Epochs Plot:



Test Set and Decision Boundary Plot:



Region above the line is +1. Region below the line is -1.

Error Rate:

.0636

Iteration 2:

Net Construction:

The last implementation got us very close to a perfect set, or at least as perfect as we can get with the architecture taught in the course so far. However, our inputs are decimal and applying the bipolar activation function right away loses any information that we might get by scaling. For this iteration, we will be implementing the extended delta rule with a sigmoid activation function.

Activation function:

This activation function will be taking a decimal value as the input, then applying the bipolar sigmoid formula. It will also take a sigma value for the formula, which will be defined in the main script. It will then output the value after the formula has been applied. Note: I had issues for some reason with using the normal bipolar sigmoid formula. I am not sure why this happened, but the issues disappeared when I used the binary function and changed the bounds. I graphed out the two and got similar results.

```
function [output] = sigmoid_act(yin,sigma)

        output = (1./(1 + exp(-sigma .* yin)))* 2 - 1;
end
```

Since we are using the extended delta rule, we also need a function based on the derivative of the sigmoid.

```
function [output] = sigmoiddiff_act(yin,sigma)

      output = (sigma/2)*(1 + sigmoid_act(yin,sigma))*(1 - sigmoid_act(yin,sigma));
end
```

Algorithm:

Most steps are similar to the first iteration, so I will go over the changes. First, we will define the sigma at .25. I found this value to be optimal in regards to the overall error rate.

```
sigma = .25;
```

In the training phase, we will apply the sigmoid activation function instead of the bipolar activation function before calculating the error. This will preserve scaling in the error calculation. We will also update the weights using the extended delta rule, adding in the term for the derivative.

```
for x = epochs
    for n = 1:length(S(1,:))
        a = alpha(iter);
        yin = W.' * S(:,n);
        yin = sigmoid_act(yin,sigma);
        err = t(n) - yin;
        errsq = err^2;
        totalerrsq = totalerrsq + errsq;
        W = W + a * err * S(:,n) * sigmoiddiff_act(yin,sigma);
        iter = iter + 1;
    end
    MSE(x) = totalerrsq/(x*n);
end
```

In the matching phase, we will apply the sigmoid activation function again to get our youts. Then, we will have to apply the bipolar activation function again in order to get values that match up with our target set. I do not think this is the best way to calculate the matches. However, I was not able to think of an alternative method.

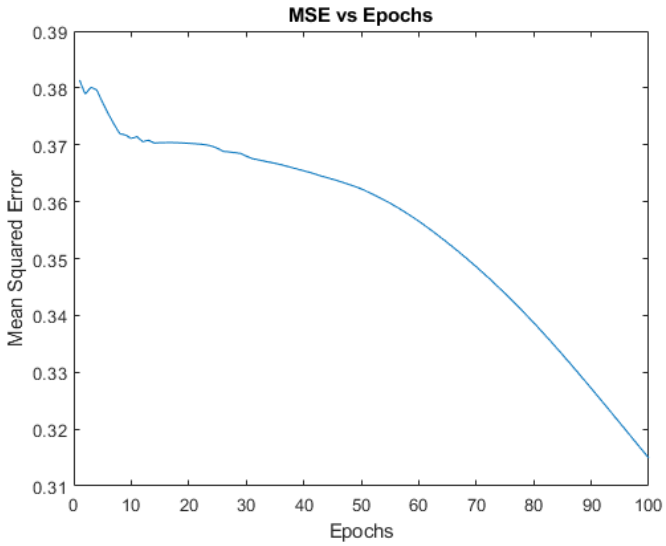
```
for n = 1:length(STest(1,:))
    yout = W.' * STest(:,n);
    yout = sigmoid_act(yout,sigma);
    yout = bipolar_act(yout);
    if yout == tTest(n)
        match(n) = 1;
    end
end
```

Final Weights:

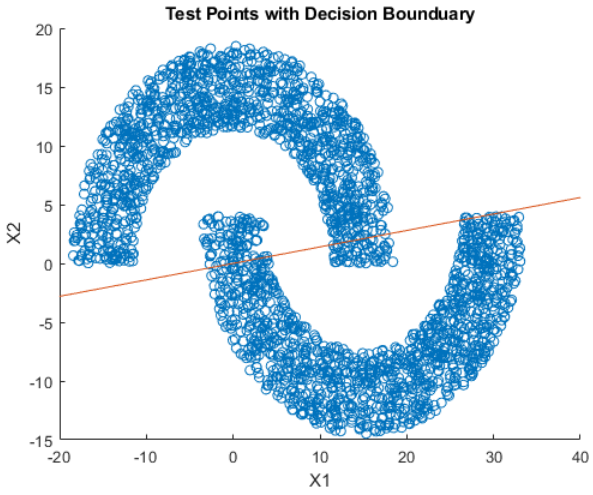
W = [- .03318;

2.3691];

MSE vs Epochs Plot:



Test Set and Decision Boundary Plot:



Region above the line is +1. Region below the line is -1.

Error Rate:

.0620

Analysis and Future Steps:

There is a decrease of .0016 in the error rate when introducing scaling. It is not a huge margin, but it is consistent enough to call it a boost in accuracy. We can also see from the graph of the MSE that the error rate decreases in a much smoother fashion when using the extended delta rule, possibly guarding against over-fitting. For likely future steps, I would like to explore the possibility of applying iterative conditions to the inputs of the test set. This would take a form similar to the Hopfield net algorithm. In addition, I see from the decision boundary plots that there is no way to get perfect classification in this

problem using a linear boundary. There is always going to be a lot of crosstalk in our solution due to the overlap of the testing sets. Therefore, a multilayer net may be better suited for this problem.