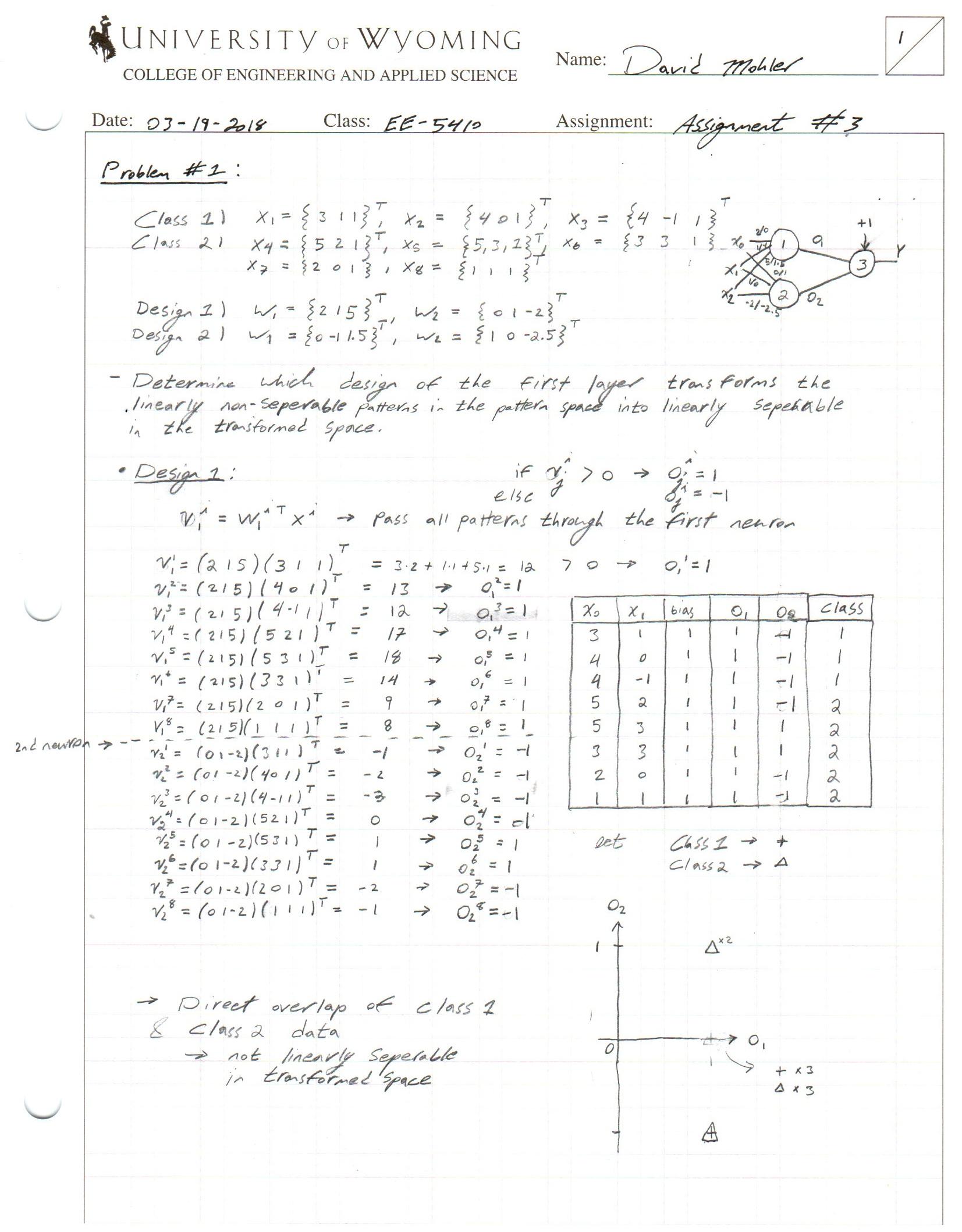
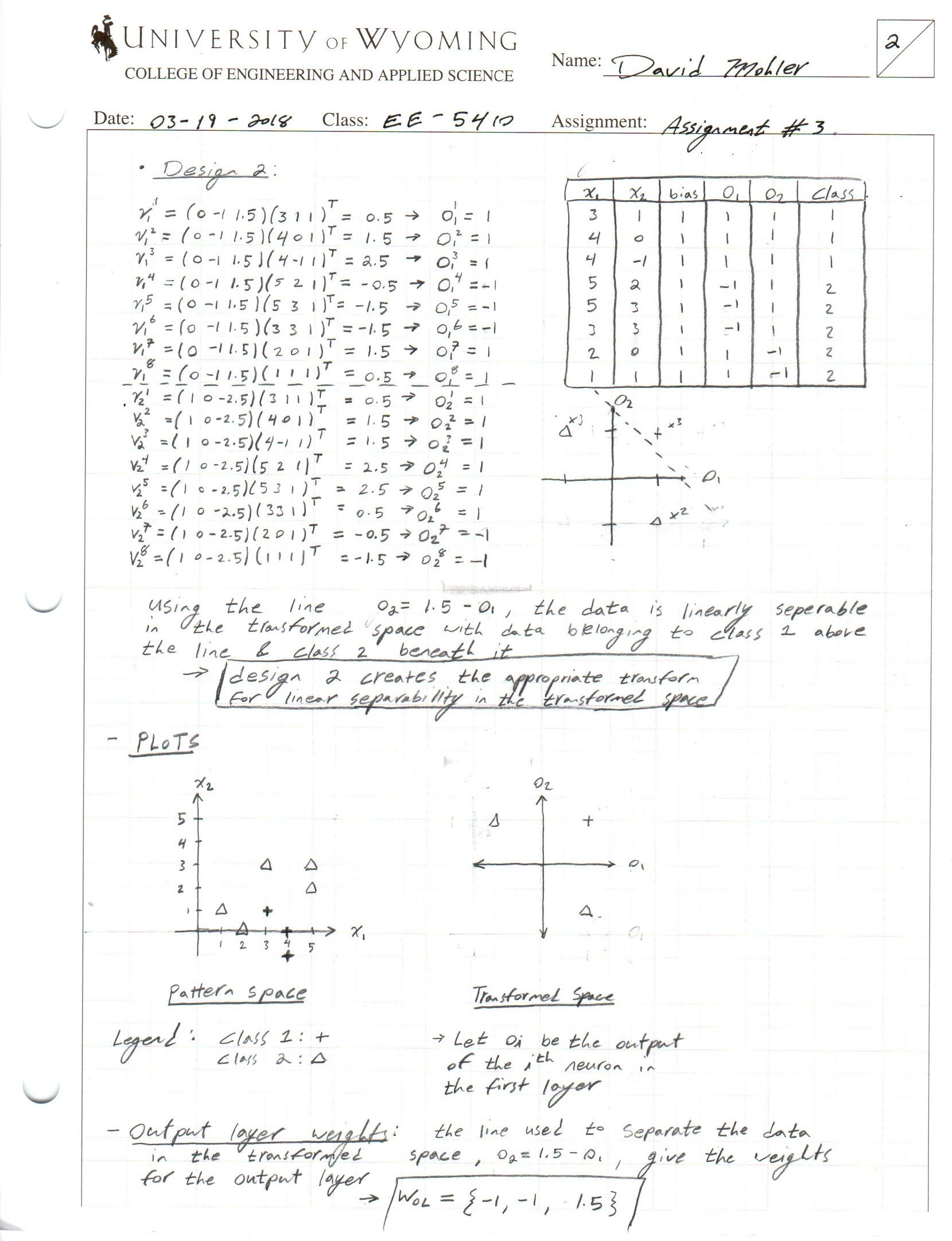
**David R Mohler**

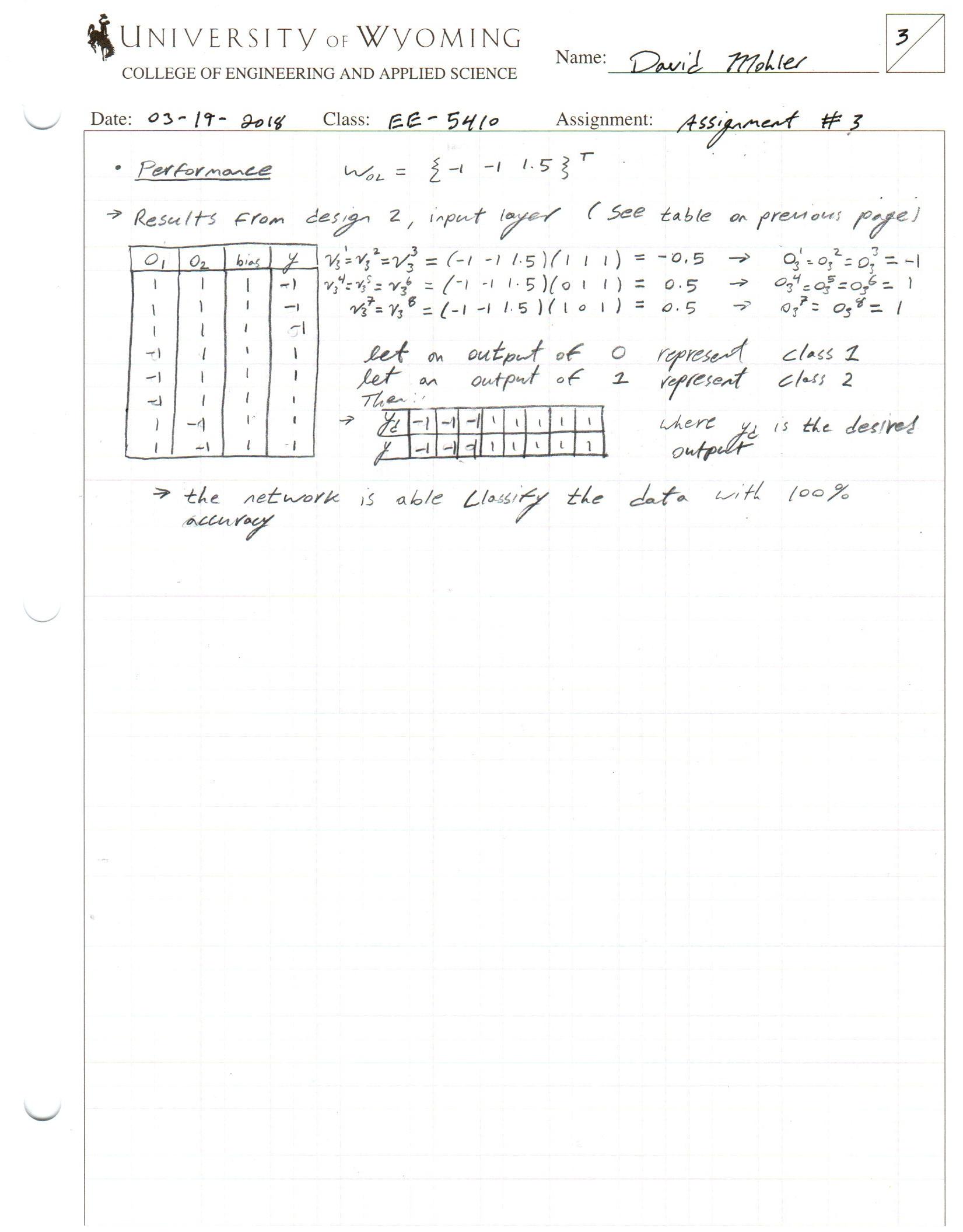
**EE 5410: Neural Networks**

**Assignment #3**

**03-23-2018**

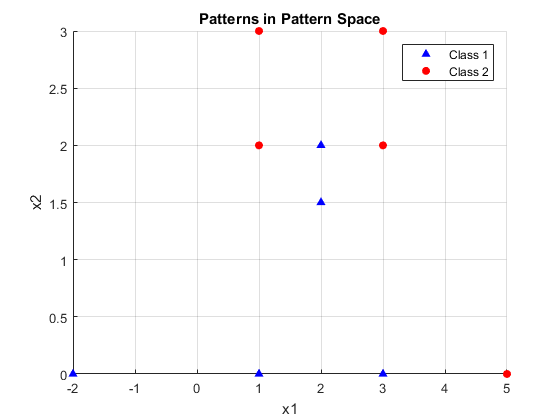
****

****

****

**-1**

**Problem #2**

**\*NOTE: Most calculations in problem 2 are completed in the attached Matlab code. \***

|  |  |  |
| --- | --- | --- |
| **X1** | **X2** | **Class** |
| **1** | **3** | **2** |
| **3** | **3** | **2** |
| **1** | **2** | **2** |
| **2** | **2** | **1** |
| **3** | **2** | **2** |
| **2** | **1.5** | **1** |
| **-2** | **0** | **1** |
| **1** | **0** | **1** |
| **3** | **0** | **1** |
| **5** | **0** | **2** |

*Input Data Patterns and Associated Classes Input Patterns in the pattern space*

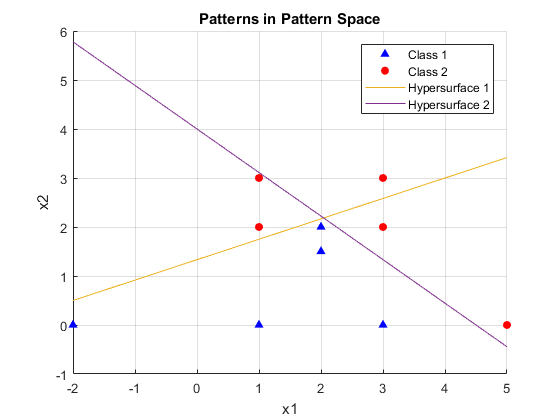
* From examination of the plotted data it is possible to classify the data appropriately with two linear hypersurfaces such that all the data in Class 1 will lie below the logical intersection of both curves.
* Using the parametric approach, I select a series of two candidate points in the pattern space from which to construct each respective decision boundary.
  + For the first hypersurface, let the analytically chosen points p1 and p2 be:

Using the standard equation of a line given two points, , we are able to obtain the equation of the first linear hypersurface as:

* + - .

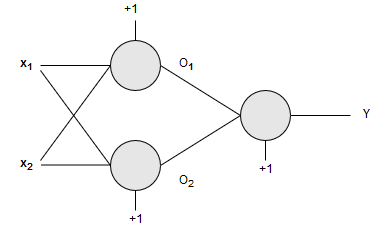
Similarly, it is possible to obtain the second hypersurface equation using the candidate points:

From these candidate points the second hypersurface equation can be represented as:

* ****The two hypersurface can then be superimposed in the pattern space as shown below:

*Hypersurfaces and Input Patterns in Pattern Space*

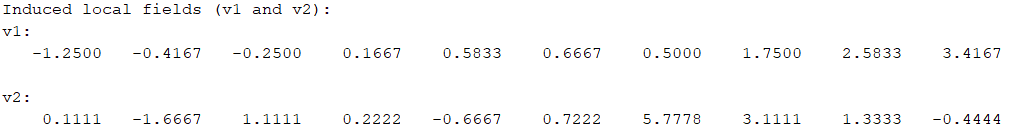
* These two surfaces dictate the number of neurons used in the first layer of the network. This gives us an over a three perceptron network, with the first two in the input layer and a single perceptron as the output layer to appropriately classify a two class problem.

******

*Neural Network Diagram*

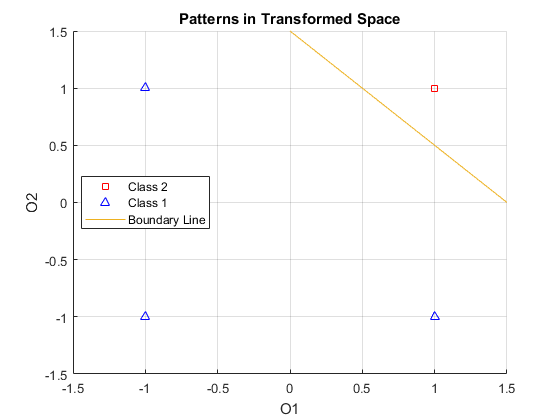
* Using the analytically generated hyper surface we extract the coefficients from the hypersurface equations, these describe the respective weight vectors for each neuron in the input layer
* Using the input data augmented with a bias of positive 1, we next begin to compute the induced local field for each pattern presented to the network. The following equation can be used to calculate the Net, where is the net related to the pattern presented to the neuron. Following this the Heaviside activation function is applied to generate a bipolar output of each neuron.

  + Heaviside activation function



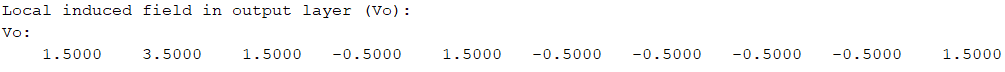
* The results of passing all the patterns through the local induced field and subsequently the activation function are summarized in the table below, where O1 and O2 are the outputs of the input layer neurons:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **X1** | **X2** | **Bias** | **O1** | **O2** | **Class** |
| **1** | **3** | **1** | **-1** | **1** | **2** |
| **3** | **3** | **1** | **-1** | **-1** | **2** |
| **1** | **2** | **1** | **-1** | **1** | **2** |
| **2** | **2** | **1** | **1** | **1** | **1** |
| **3** | **2** | **1** | **1** | **-1** | **2** |
| **2** | **1.5** | **1** | **1** | **1** | **1** |
| **-2** | **0** | **1** | **1** | **1** | **1** |
| **1** | **0** | **1** | **1** | **1** | **1** |
| **3** | **0** | **1** | **1** | **1** | **1** |
| **5** | **0** | **1** | **1** | **-1** | **2** |



*Patterns transformed through first layer Plot of patterns in the transformed space*

* As shown in the figure above, the chosen weight vectors have enabled the first network layer to transform the patterns such that they are linearly separable. In order to show this, we again select candidate points in the transform space and develop the boundary hypersurface. For the line show we have used the points , which yields the hypersurface:
* Similar to the method for the first layer we then extract the weight vector for the single neuron in the output layer from the hypersurface equation, this gives:
* Lastly we are able to test the performance of the network by passing the transformed patterns through the output layer using the same calculation of the local induced field and application of the Heaviside activation function to ensure that the network correctly classifies all 10 data points. The results of the net calculations are shown below.



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **X1** | **X2** | **Bias** | **O1** | **O2** |  |  |
| **1** | **3** | **1** | **-1** | **1** | **2** | **2** |
| **3** | **3** | **1** | **-1** | **-1** | **2** | **2** |
| **1** | **2** | **1** | **-1** | **1** | **2** | **2** |
| **2** | **2** | **1** | **1** | **1** | **1** | **1** |
| **3** | **2** | **1** | **1** | **-1** | **2** | **2** |
| **2** | **1.5** | **1** | **1** | **1** | **1** | **1** |
| **-2** | **0** | **1** | **1** | **1** | **1** | **1** |
| **1** | **0** | **1** | **1** | **1** | **1** | **1** |
| **3** | **0** | **1** | **1** | **1** | **1** | **1** |
| **5** | **0** | **1** | **1** | **-1** | **2** | **2** |

* From the net calculations we apply the activation function, the results of the bipolarization are shown in the table below. Note that is the desired class label and is the output of the neural network, also note that while the Bipolar Heaviside function provides outputs of -1 or 1, the class labels have been shown in human-friendly notation where a -1 from the network represents class 1 and a 1 represents the second class.

*Results of parametric perceptron network*

* From the table it can be seen that the parametrically designed network was able to correctly classify all ten of the given data patterns. **(Please see attached Matlab for code used to perform calculation and plotting) .**

**Problem #3:**

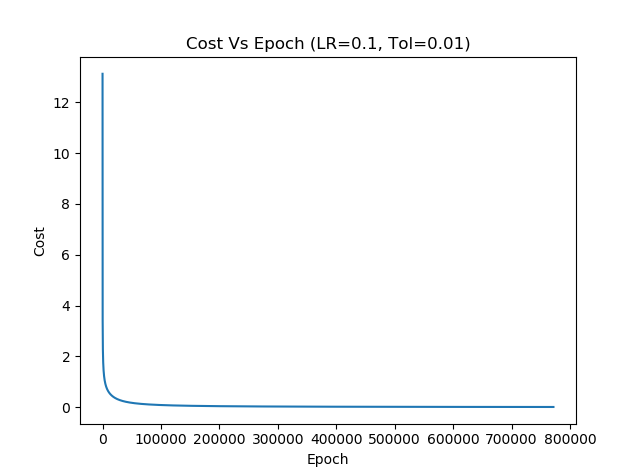
* For the continuous perceptron we no longer use the discrete Heaviside activation function. Instead, we now must use a continuous activation function. Given that our training data is classified with desired outputs of either zero or one, an appropriate candidate for the activation function would be the Sigmoid function. The sigmoid function and its derivative can be respectively described using the equations below, where is the local induced field of a given pattern.
  + ,
* When applying the delta learning rule we apply the cost function based upon the square of the error as described below, where represents the number of patterns and represents the weight parameters:
* The associated gradient of this cost function is:

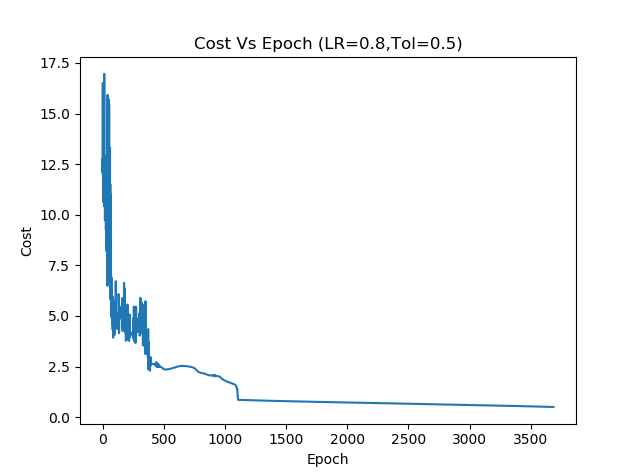
**Table 1 Perceptron Training Results**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Training** | **Vector of Weights (Initial)** | | | | **Vector of Weights (Final)** | | | | **Learn rate** | **Tol** | **# of Epochs** |
| **w1** | **w2** | **w3** | **w0** | **w1** | **w2** | **w3** | **w0** |  |
| **T1** | 0.170 | 0.115 | 0.655 | 0 | 41.2 | 65.8 | -19.2 | -82.2 | 0.1 | 0.01 | 771,720 |
| **T2** | 0.949 | 0.008 | 0.208 | 0 | 25.08 | 39.88 | -11.7 | -50.2 | 0.1 | 0.1 | 86,581 |
| **T3** | 0.45 | 0.554 | 0.565 | 0 | 15.00 | 23.90 | -6.94 | -29.8 | 0.25 | 0.5 | 6961 |
| **T4** | 0.132 | 0.536 | 0.566 | 0 | 23.13 | 36.02 | -9.98 | -43.6 | 0.8 | 0.5 | 3687 |
| **T5** | 0.204 | 0.255 | 0.591 | 0 | 8.67 | 20.87 | -4.24 | -6.33 | 2.0 | 5.0 | 178 |

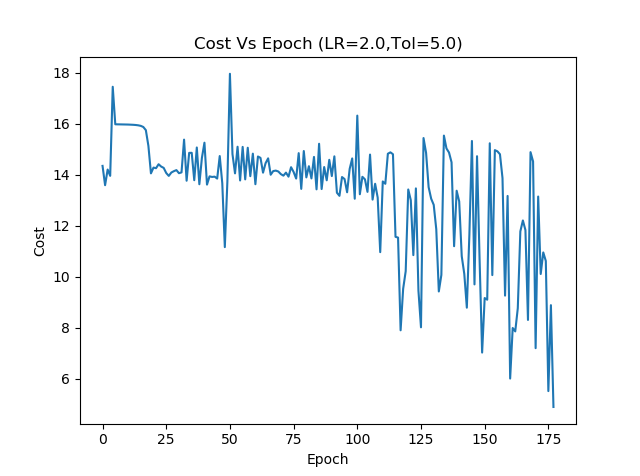
***Table 2: Classification of Test Data***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sample | x1 | x2 | x3 | YP (T1) | YP (T2) | YP (T3) | YP (T4) | YP (T5) |
| 1 | -0.3665 | 0.062 | 5.9891 | 0 | 0 | 0 | 0 | 0 |
| 2 | -0.7842 | 1.1267 | 5.5912 | 1 | 1 | 1 | 1 | 0 |
| 3 | 0.3012 | 0.5611 | 5.8234 | 1 | 1 | 1 | 1 | 0 |
| 4 | 0.7757 | 1.0648 | 8.0677 | 1 | 1 | 1 | 1 | 1 |
| 5 | 0.157 | 0.8028 | 6.304 | 1 | 1 | 1 | 1 | 0 |
| 6 | -0.7014 | 1.0316 | 3.6005 | 1 | 1 | 1 | 1 | 1 |
| 7 | 0.3748 | 0.1536 | 6.1537 | 0 | 0 | 0 | 0 | 0 |
| 8 | -0.692 | 0.9404 | 4.4058 | 1 | 1 | 1 | 1 | 1 |
| 9 | -1.397 | 0.7141 | 4.9263 | 0 | 0 | 0 | 0 | 0 |
| 10 | -1.8842 | -0.2805 | 1.2548 | 0 | 0 | 0 | 0 | 0 |

* **Analysis:** Over the five trials we observed the effect of varying the learning rate and error tolerance of the network on the ability of the network to appropriately classify the data. From table 1 we can see that in order to achieve a small square error across all training patterns (i.e. that the number of epochs required to accomplish this is rather large at 771,720 epochs. With this training it was able to classify the test data with 100% accuracy (assignment 2 used as a benchmark for appropriate classifications of test data). However, if we also analyze the cost vs the epoch we can see that the network spends hundreds of thousands of epochs gaining very little progress.

 Instead, the same classification results were obtained using a less stringent tolerances and higher learning rates for training of the network. As shown, we are able to achieve 100% classification of the 10 test patterns with a learning rate of 0.8 and a tolerance of 0.5. This reduced the number of epoch necessary for training to 3,687. While the performance of the training is of low priority, the reduction in epochs and repeated exposure to the patterns with little change in cost has the benefit of reducing the likelihood that the network is saturated and no longer learning. Even though the cost is not as monotonic in nature as the initial trial, the envelope has a generic trend toward zero. For the trials conducted, this is the best trained network, as it is capable of perfect classification, as well as the lowest amount of time dedicated to training. The results of this trial are shown below:

In order to observe some of the limitations of network, we tested the performance with a much looser tolerance of 5.0 and large learning rate at 2.0. While the number of epochs necessary to train the network were significantly reduced (178), we see that the network failed to classify the data perfectly, with a success of only 70%.



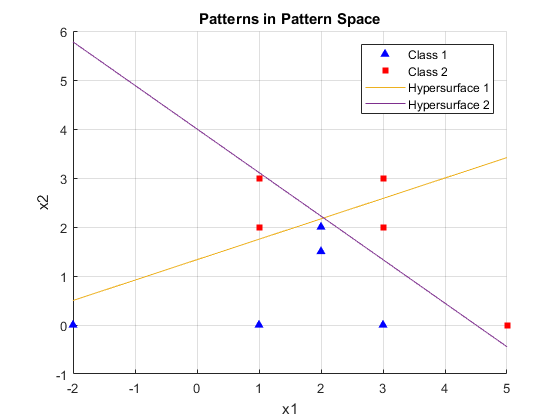
***Table 3: Resultant Hypersurface Equations***

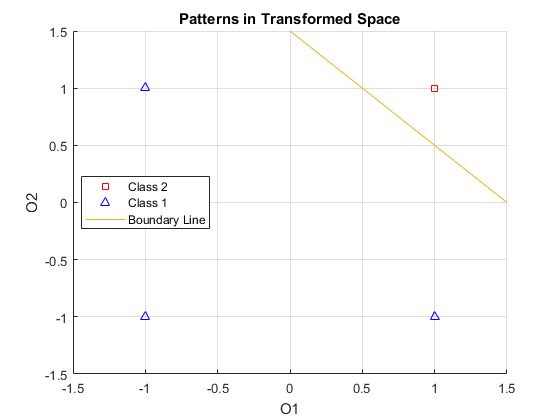
|  |  |
| --- | --- |
| **Training** | **Hypersurface Equation** |
| **T1** |  |
| **T2** |  |
| **T3** |  |
| **T4** |  |
| **T5** |  |

* **Comparison:** In assignment two, the discrete perceptron was able to complete the training of the network in an average of approximately 400 epochs. With this training, the discrete perceptron was able to classify the test data patterns with 100% accuracy regardless of the initial weight conditions. This is largely due to the “search” style of training that was performed by the network. In contrast, the continuous perceptron was not able to compete in terms of necessary epochs for training. In our best case of continuous training we succeeded in correctly classifying the test data after 3,687 training epochs. In order to achieve similar or lower values of training epochs the continuous system lost the accuracy to ensure convergence to a completely correct classification. For this task, it appears that the discrete perceptron is a better candidate in terms of efficiency and for correct classification of the testing patterns.

%David R Mohler  
%EE5410: Neural Nets  
%HW1: Exhibition of Data Classification (prob 3)  
clear  
close all  
dataSorted = [2 2; 2 1.5; -2 0; 1 0; 3 0; %Class 1: x4,6, 7, 8, 9  
 1 3; 3 3; 1 2; 3 2; 5 0]; %Class 2: 1 2 3 5 10  
  
data = [1 3; 3 3; 1 2; 2 2; 3 2;  
 2 1.5; -2 0; 1 0; 3 0; 5 0;]; %data in order from 1-10  
  
bias = ones(10,1);  
aug = [data bias];  
  
%Decision Boundary (parametric approach)  
x1 = linspace(-2,5);  
%Points used to generate 1st boundary line  
p1 = [-2,0.5];  
p2 = [4,3];  
%Points used to generate 2nd boundary line  
p3 = [4.5,0];  
p4 = [0,4];  
  
slope1 = ((p2(2)-p1(2))/(p2(1)-p1(1)));  
slope2 = ((p4(2)-p3(2))/(p4(1)-p3(1)));  
b1 = (slope1\*-p1(1))+p1(2);  
b2 = (slope2\*-p3(1))+p3(2);  
bound1 = slope1\*x1+b1;  
bound2 = slope2\*x1+b2;  
  
%Extract neuron weights from hypersurface lines  
W1 = [slope1 -1 b1]';  
W2 = [slope2 -1 b2]';  
  
%Calculate Net of each neuron and use heaviside actiavtion to  
%assign neuron outputs  
for i = 1:10  
 v1(i) =dot(W1,aug(i,:));  
 v2(i) =dot(W2,aug(i,:));  
 %Heaviside Neuron 1 output  
 if v1(i) > 0  
 O1(i)=1;  
 else  
 O1(i)=-1;  
 end  
 %Heaviside Neuron 2 output  
 if v2(i) > 0  
 O2(i)=1;  
 else  
 O2(i)=-1;  
 end  
end  
disp('Input data patterns:')  
disp(data)  
txt = 'Induced local fields (v1 and v2):';  
disp(txt)  
disp('v1: ')  
disp(v1)  
disp('v2:')  
disp(v2)  
txt = 'Input layer results(O1 and O2):';  
disp(txt)  
disp('O1: ') %increment for human friendly classifications  
disp(O1)  
disp('O2: ')  
disp(O2)  
  
%Using points [1.5,0] and [0,1.5] establsih linear seperability between  
%classes for output layer and extract weights from chosen boundary  
xOL = linspace(-0.5,1.5);  
boundOL = 1.5-xOL; %Boundary line in transform space  
WOL = [-1 -1 1.5]';%Output neuron weights  
  
O = [O1' O2' bias];  
for i = 1:10  
 vo(i) =dot(WOL,O(i,:));  
 %Heaviside Neuron 1 output  
 if vo(i) > 0  
 y(i)=1;  
 else  
 y(i)=-1;  
 end  
end  
disp('Local induced field in output layer (Vo):')  
disp('Vo:')  
disp(vo)  
txt = 'Network output classifications (y):';  
disp(txt)  
disp(y)  
figure(1)  
hold on  
grid on  
%Class 1 data  
scatter(dataSorted(1:5,1),dataSorted(1:5,2),'^','b','filled')  
%Class 2 Data  
scatter(dataSorted(6:end,1),dataSorted(6:end,2),'s','r','filled')  
plot(x1,bound1)  
plot(x1,bound2)  
xlim([-2 5])  
legend('Class 1','Class 2','Hypersurface 1','Hypersurface 2')  
title('Patterns in Pattern Space')  
xlabel('x1')  
ylabel('x2')  
  
figure(2)  
hold on  
grid on  
gscatter(O1(:),O2(:),y,'rb','s^')  
% scatter(O1(i),O2(i),'^','b','filled')  
% scatter(O1(i),O2(i),'r','filled')  
plot(xOL,boundOL)  
xlim([-1.5 1.5])  
ylim([-1.5 1.5])  
legend('Class 2','Class 1','Boundary Line')  
title('Patterns in Transformed Space')  
xlabel('O1')  
ylabel('O2')

Input data patterns:  
 1.0000 3.0000  
 3.0000 3.0000  
 1.0000 2.0000  
 2.0000 2.0000  
 3.0000 2.0000  
 2.0000 1.5000  
 -2.0000 0  
 1.0000 0  
 3.0000 0  
 5.0000 0  
  
Induced local fields (v1 and v2):  
v1:   
 Columns 1 through 7  
  
 -1.2500 -0.4167 -0.2500 0.1667 0.5833 0.6667 0.5000  
  
 Columns 8 through 10  
  
 1.7500 2.5833 3.4167  
  
v2:  
 Columns 1 through 7  
  
 0.1111 -1.6667 1.1111 0.2222 -0.6667 0.7222 5.7778  
  
 Columns 8 through 10  
  
 3.1111 1.3333 -0.4444  
  
Input layer results(O1 and O2):  
O1:   
 -1 -1 -1 1 1 1 1 1 1 1  
  
O2:   
 1 -1 1 1 -1 1 1 1 1 -1  
  
Local induced field in output layer (Vo):  
Vo:  
 Columns 1 through 7  
  
 1.5000 3.5000 1.5000 -0.5000 1.5000 -0.5000 -0.5000  
  
 Columns 8 through 10  
  
 -0.5000 -0.5000 1.5000  
  
Network output classifications (y):  
 1 1 1 -1 1 -1 -1 -1 -1 1





[*Published with MATLAB® R2017a*](http://www.mathworks.com/products/matlab)