# Laboratory Assignment #2: Pattern Recognition via Percepton

Bryan Guner, James Martinez, and Olivia Shanley
Anthony Deese, Ph. D.
Artificial Neural Networks - ELC 470-01
The College of New Jersey
Department of Electrical and Computer Engineering (ECE)
02/27/2017

**Introduction and Procedure**

For this lab, we trained a neural network to recognize hand-drawn symbols based off of the results gathered by in-class testing. This began with defining three symbols (a square, a triangle, and an 'X' shape) and having peers draw each symbol over the six sensors as they felt best fit them. We then proceeded to use the collected training data to create a truth table for each symbol, noting which sensors were "on" (used in the drawing of the symbol). This data was then fed to the neural network, with initial weights for each sensor set. The sensors that we expected to not be activated for each symbol were given a weight of -1 and the remainder were given a collective weight of one distributed over each sensor. For example, if we expected sensors 1, 3, 5, and 6 to be activated for a square, they each were initialized to a weight of .25 while sensors 2 and 4 were initialized to a weight of -1.

**Results**

As can be seen from the following graphs, both the 'X' and triangle had relatively low, sporadic performances. The losses per iteration seemed to oscillate wildly, perhaps because of their training data had more inconsistencies. The square, however, performed best, converging after roughly 2700 iterations to an error of 0.0%. This is likely due to the near perfect training data collected from the peer trials.

**Graphs**

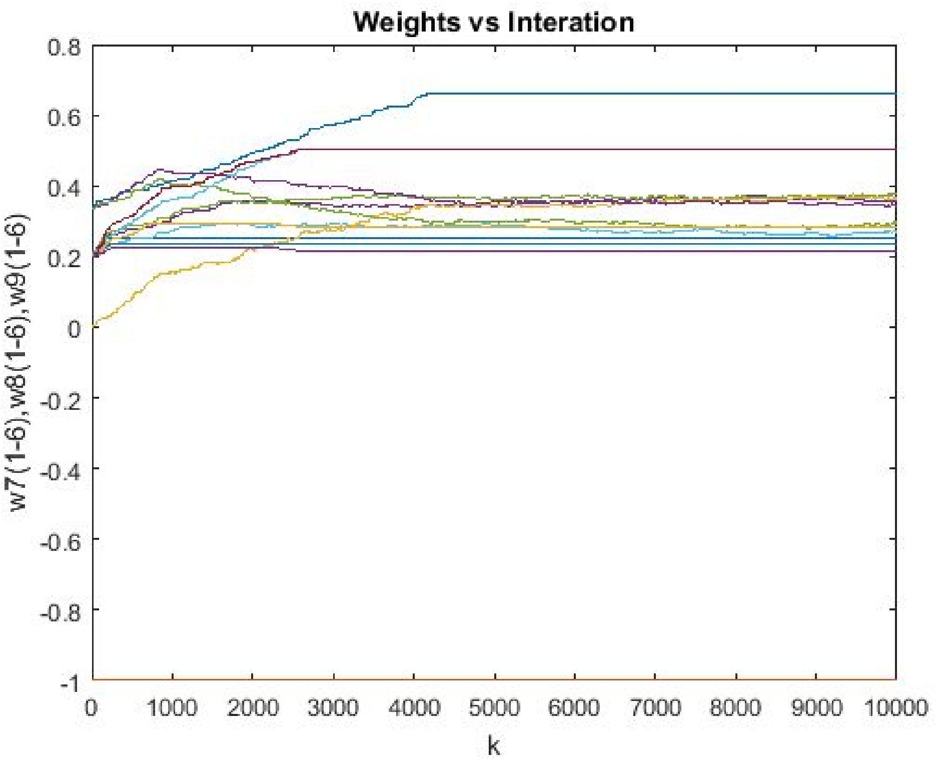

Figure 1. Changes in Weights per Iteration

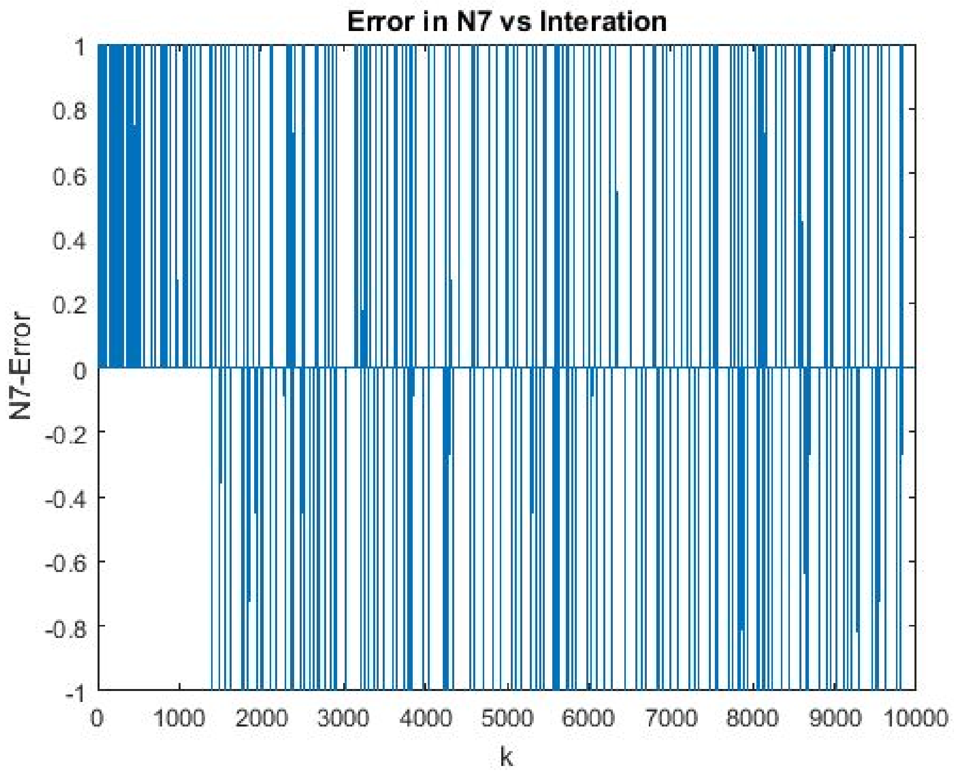

Figure 2. Error in Node 7 (‘X’ Sensor) per Iteration

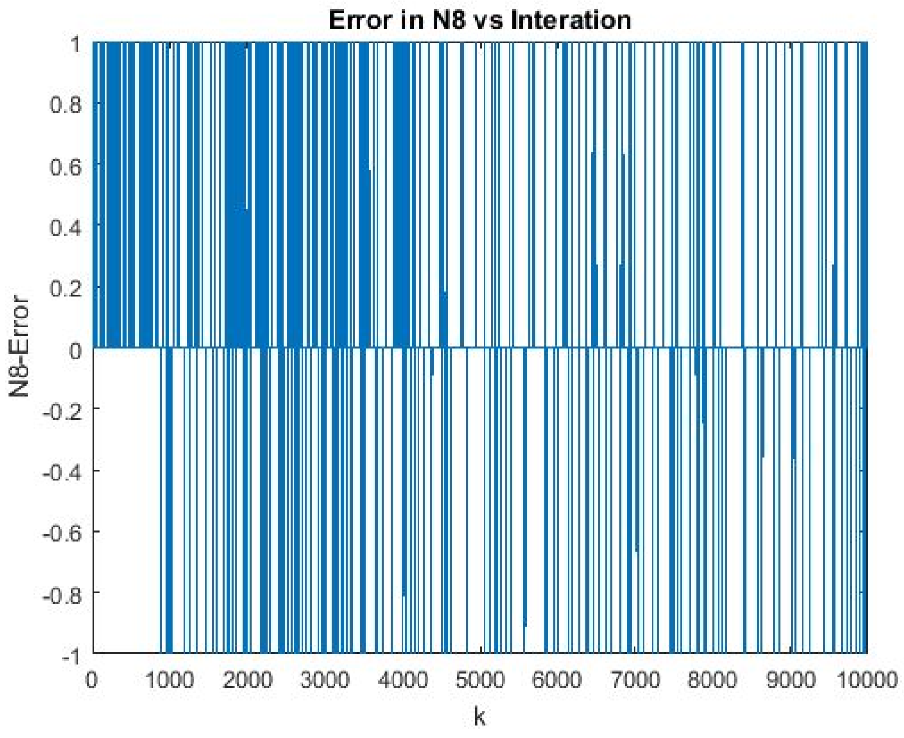

Figure 3. Error in Node 8 (Triangle Sensor) per Iteration

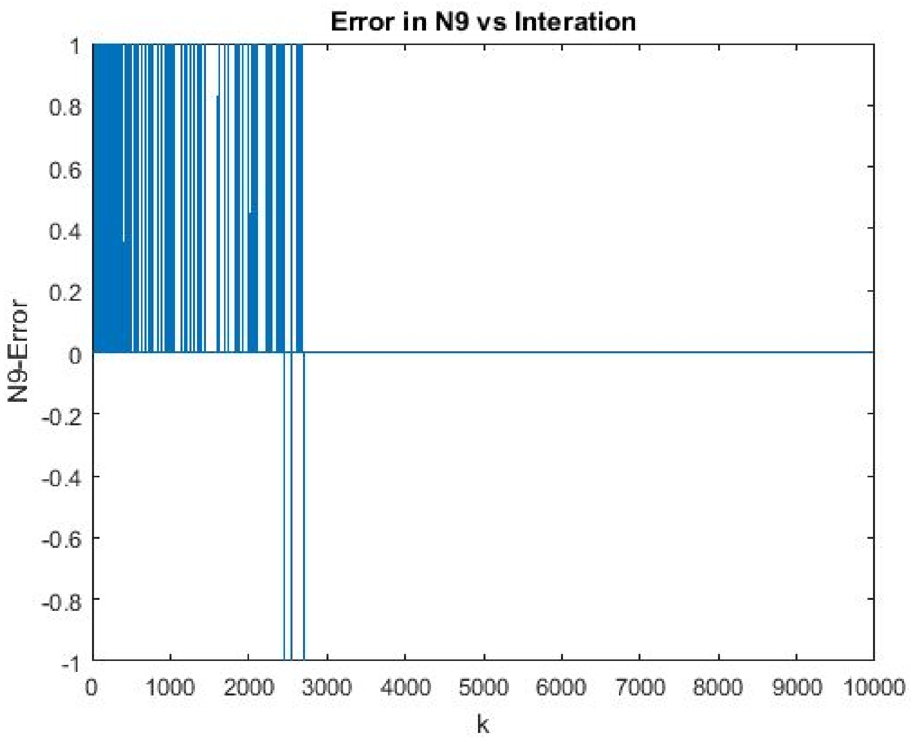

Figure 3. Error in Node 9 (Square Sensor) per Iteration

**Conclusion**

The goal of this exercise was to develop an ANN that recognizes patterns based off of an example with six input, two neurons, and twelve synapses. The parameters of design where a structure with at least six neurons within the retina layer and exactly three neurons in the output layer. In addition this network had to contain at least 10 weighted synapse connections, our final manifestation of this structure contained 13 synapses. In order to implement our ANN our team developed simulation files in MatLab allowed us to test different structures. Our final configuration was achieved by defining a simulation with the maximum number of synapse connections (18). We then tried turning different synapses on and off within the script to find an optimal solution. We initialized the synapse weights by anticipating the sensors that were most likely to be triggered by each symbol, the sensors that we expected to not be activated for each symbol were given a weight of -1 and the remainder were given a weight of 1 distributed evenly among the selected sensors. Our design was able to accurately identify patterns, however there were errors for symbols that could easily evade the sensor network where that data was collected.

Appendix A - Matlab Code

```
%sensor configurations for each trial
x=[[1, 0, 1, 1, 1, 1];[0, 0, 0, 1, 1, 1];[1, 0, 1, 1, 1, 1];[1, 0, 1, 1, 1, 1];[1 0 1 1 1 0];[1 0 1 1 1 1];[1
0 0 1 1 1];[1 0 1 1 1 1];[1 0 1 1 1 1];[1 0 1 1 1 1];[1 0 1 1 1 1];[1 0 1 1 1 0];[0 1 0 0 1 1];[0 1 0 1
1 1];[0 1 0 0 1 1];[0 1 0 0 1 1];[0 1 0 1 1 0];[0 1 0 1 1 1];[0 1 0 1 0 0];[0 1 0 1 1 1];[0 0 0 1 1
1];[0 1 0 1 1 1];[0 1 0 0 1 1];[0 1 0 1 1 1];[1 1 1 0 1 1];[1 1 1 0 1 1];[1 1 1 0 1 1];[1 1 1 0 1 1];[1
1 1 0 1 1];[1 1 1 0 1 1];[1 1 0 0 1 0];[1 1 1 0 0 1];[1 1 0 0 0 0];[1 1 1 0 1 1];[1 1 1 0 1 0];[0 1 1 0
1 1]]; %table for x
N7_ideal=[1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]; %ideal response
for x
N8_ideal=[0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0]; %ideal response
for tri
N9_ideal=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1]; %ideal response
for sqr


w7=zeros(6,9999); %initialize
w8=zeros(6,9999);
w9=zeros(6,9999);
%initialize
%X
w7(1,1)=1/5;
w7(2,1)=-1;
w7(3,1)=1/5;
w7(4,1)=1/5;
w7(5,1)=1/5;
w7(6,1)=1/5;
%triangle
w8(1,1)=-1;
w8(3,1)=-1;
w8(2,1)=1/3;
w8(5,1)=1/3;
w8(6,1)=1/3;
%square
w9(1,1)=1/5;
w9(2,1)=1/5;
w9(3,1)=1/5;
```

```
w9(4,1)=-1;
w9(5,1)=1/5;
w9(6,1)=1/5;


 w0 = 1; %weight threshold
 sigma = .005;
 k=1; %iteration number
 t=1:1:9999; %used as x axis in plot
 N7_delt=ones(1,9999); % initialize 800 ones as the difference in y
 N8_delt=ones(1,9999);
 N9_delt=ones(1,9999);
   for k=1:1:35


N7_calc=(w7(1,k)*x(k,1)+w7(2,k)*x(k,2)+w7(3,k)*x(k,3)+w7(4,k)*x(k,4)+w7(5,k)*x(k,5)+w7(
6,k)*x(k,6));

N8_calc=(w8(1,k)*x(k,1)+w8(2,k)*x(k,2)+w8(3,k)*x(k,3)+w8(4,k)*x(k,4)+w8(5,k)*x(k,5)+w8(
6,k)*x(k,6));

N9_calc=(w9(1,k)*x(k,1)+w9(2,k)*x(k,2)+w9(3,k)*x(k,3)+w9(4,k)*x(k,4)+w9(5,k)*x(k,5)+w9(
6,k)*x(k,6));

        if (N7_calc >=w0) %unit step function with threshold at w0
              N7_calc=1;
        else
            N7_calc=0;
        end

         if (N8_calc >=w0) %unit step function with threshold at w0
              N8_calc=1;
        else
            N8_calc=0;
         end

         if (N9_calc >=w0) %unit step function with threshold at w0
              N9_calc=1;
        else
```

```
           N9_calc=0;
        end



        N7_delt(k) = N7_ideal(k) - N7_calc;
        N8_delt(k) = N8_ideal(k) - N8_calc;
        N9_delt(k) = N9_ideal(k) - N9_calc;

        for i=1:1:6
           w7(i,k+1)= rand()*sigma*N7_delt(k)*x(k,i)+w7(i,k);
           w8(i,k+1)= rand()*sigma*N8_delt(k)*x(k,i)+w8(i,k);
           w9(i,k+1)= rand()*sigma*N9_delt(k)*x(k,i)+w9(i,k);
        end
    end
% % % % % % % % % %
   while (k<10000) %randomize the order of the combinations
        j= randi([1 36]);


N7_calc=(w7(1,k)*x(j,1)+w7(2,k)*x(j,2)+w7(3,k)*x(j,3)+w7(4,k)*x(j,4)+w7(5,k)*x(j,5)+w7(6,
k)*x(j,6));

N8_calc=(w8(1,k)*x(j,1)+w8(2,k)*x(j,2)+w8(3,k)*x(j,3)+w8(4,k)*x(j,4)+w8(5,k)*x(j,5)+w8(6,
k)*x(j,6));

N9_calc=(w9(1,k)*x(j,1)+w9(2,k)*x(j,2)+w9(3,k)*x(j,3)+w9(4,k)*x(j,4)+w9(5,k)*x(j,5)+w9(6,
k)*x(j,6));

        if (N7_calc >=w0) %unit step function with threshold at w0
            N7_calc=1;
        else
           N7_calc=0;
        end

         if (N8_calc >=w0) %unit step function with threshold at w0
            N8_calc=1;
        else
           N8_calc=0;
         end
         if (N9_calc >=w0) %unit step function with threshold at w0
              N9_calc=1;
        else
           N9_calc=0;
        end

        N7_delt(k) = N7_ideal(j) - N7_calc;
        N8_delt(k) = N8_ideal(j) - N8_calc;
        N9_delt(k) = N9_ideal(j) - N9_calc;

        for i=1:1:6
           w7(i,k+1)= rand()*sigma*N7_delt(k)*x(j,i)+w7(i,k);
           w8(i,k+1)= rand()*sigma*N8_delt(k)*x(j,i)+w8(i,k);
           w9(i,k+1)= rand()*sigma*N9_delt(k)*x(j,i)+w9(i,k);
        end
        k=k+1;
    end
% % % % % % % % % %
 figure(1); %plot each weight on the same graph
 plot(w7(1,t)); hold on;
 plot(w7(2,t)); hold on;
 plot(w7(3,t)); hold on;
 plot(w7(4,t)); hold on;
 plot(w7(5,t)); hold on;
 plot(w7(6,t)); hold on;
 plot(w8(1,t)); hold on;
 plot(w8(2,t)); hold on;
 plot(w8(3,t)); hold on;
 plot(w8(4,t)); hold on;
 plot(w8(5,t)); hold on;
 plot(w8(6,t)); hold on;
 plot(w9(1,t)); hold on;
 plot(w9(2,t)); hold on;
 plot(w9(3,t)); hold on;
 plot(w9(4,t)); hold on;
 plot(w9(5,t)); hold on;
 plot(w9(6,t));
 title('Weights vs Interation')
 xlabel('k')
 ylabel('w7(1-6),w8(1-6),w9(1-6)')
% % % % % % %
 figure(2); %plot the difference in N7 X
 plot(t,N7_delt);
 title('Error in N7 vs Interation')
 xlabel('k')
 ylabel('N7-Error')

 figure(3); %plot the difference in N8 triangle
 plot(t,N8_delt);
 title('Error in N8 vs Interation')
 xlabel('k')
 ylabel('N8-Error')

 figure(4); %plot the difference in N9 square
 plot(t,N9_delt);
 title('Error in N9 vs Interation')
 xlabel('k')
 ylabel('N9-Error')
```