Jeff Gray
02.15.2016
NESC 5330
Github:jeffreygray/nesc5330/tree/master/lab4

Lab 4

Note: I conceptually discussed this material with Berk Ekmekci :)

Problem One: 4.1.1

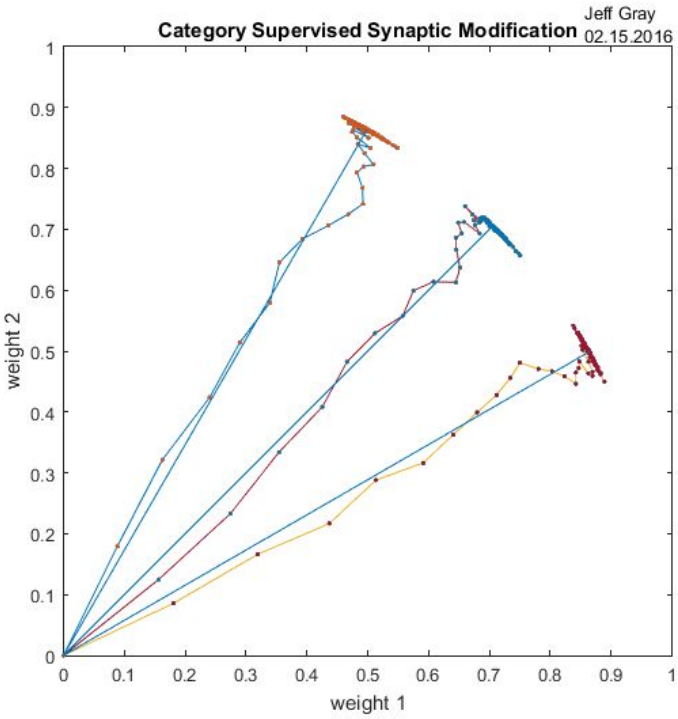

Figure 1
This plot represents the convergence of 3 data sets towards their own weight vector after 100 iterations of function catMod. Starting at the origin, each set of points gradually travels to the perimeter of the unit circle as it approaches the intersection of the weight vector's radian value. The points that conform to the edge of this circle represent convergence.

After 100 iterations, each weight value initialized at the origin converges to the radian value associated with its theoretical weight. Each iteration thus increases the precision of this estimated value. Function classifyVec.m takes an 2-dimensional column vector as input and outputs the category value of maximum postsynaptic excitation. I initially set up this program to output a 3-dimensional array with boolean values signifying whether outputs occurred, but this new criterion results in a more elegant output and allows for further analysis by implementing a 0 - “not found” option. As threshold increases, type II error initially decreases but then type I error increases due to overlapping ranges.

Problem Two: 4.1.2

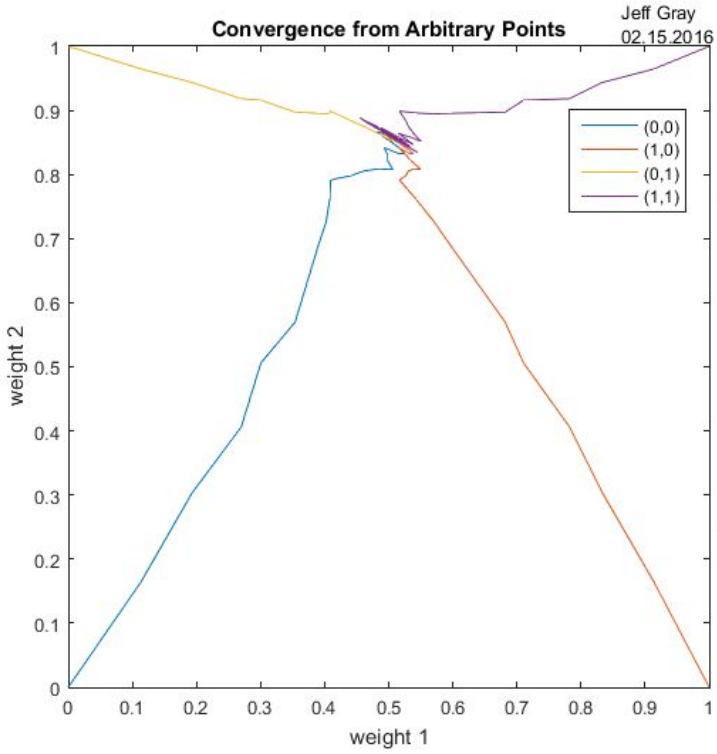

Figure 2
This plot depicts the convergence of four different weight vectors initialized at arbitrary locations (see legend). The values converge globally towards the same point in the center of the graph. The values will converge independent to their starting values.

Problem Three: 4.1.3

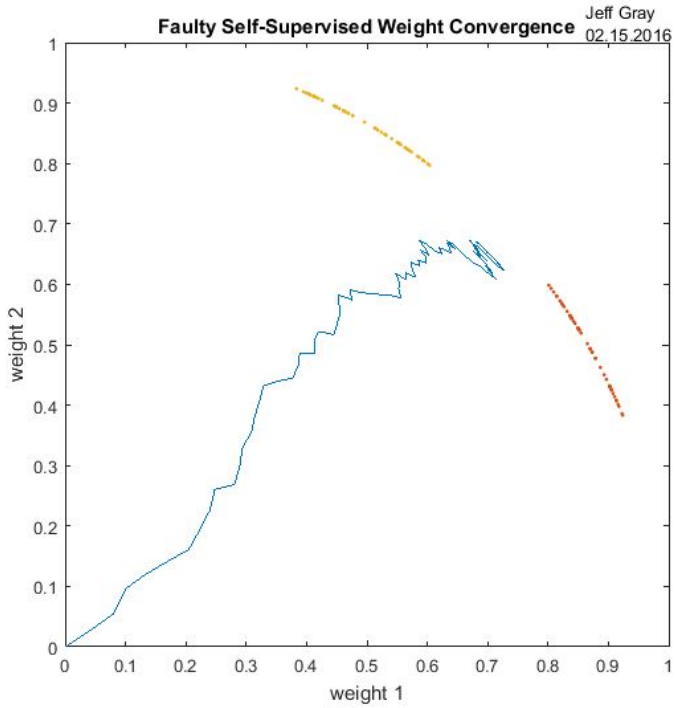

Figure 3
The orange and yellow points in this plot represent the data used as input to function catMod.
The blue line represents the outputs of catMod as they converge between the two input data sets. This is faulty because the network will not learn in this example due to the lack of two discretionary weight vectors for classification.

This current scheme cannot be used to discriminate between 2 categories. In order to do this, the programmer must impose some sort of splitting method to be able to classify the clusters that exist within the data set. In the case of the self-modified rule, this allows for small changes to be made without the presence of the category. This is due to the multiplication of the epsilon value by $y_i(t)$, which allows the network to behave in this way.

Problem Four:

Figure 2 shows global convergence when post excitation is linear. In this instance, Z_j serves as a boolean switch that only learns when active. Since the initial value rested at the origin and the categories were spaced equally, it failed to learn. However, if the initial value rested closer to either of the data clusters, it could result in classification. Note that this classification would still be false.

The initial value can detract from data presentation if it rests in the middle of data concentrations and fails to classify. This can be fixed by implementing a clustering method as previously explained.

Code

View on Github:jeffreygray/nesc5330/tree/master/lab4

```
%   jeff gray
%   jhg7nm
%   02.15.2016
%   lab4
%   file: problem 4.1.1
function [output] =  problemOne
    clc;
    clear;
    rng(9711963);

    %   making matricies that agree with other functions
        %   centered at pi/6
    a = rand(100, 1) * pi/12 + pi/6 - pi/24;
    matA = ones(3, 100);
    for i = 1 : 100
        matA(1, i) = 1;
        matA(2, i) = cos(a(i,1));
        matA(3, i) = sin(a(i,1));
    end
        %   centered at pi/4
    b = rand(100, 1) * pi/12 + pi/4 - pi/24;
    matB = ones(3, 100);
    for i = 1 : 100
        matB(1, i) = 1;
        matB(2, i) = cos(b(i,1));
        matB(3, i) = sin(b(i,1));
    end
        %   centered at pi/3
    c = rand(100, 1) * pi/12 + pi/3 - pi/24;
    matC = ones(3, 100);
    for i = 1 : 100
        matC(1, i) = 1;
        matC(2, i) = cos(c(i,1));
        matC(3, i) = sin(c(i,1));
    end

    %   plotting
    graphA = catMod(matA)';
    graphB = catMod(matB)';
    graphC = catMod(matC)';
    plot(graphA(:,1), graphA(:,2), '.')
    hold on
    plot(graphB(:,1), graphB(:,2), '.')
    hold on
    plot(graphC(:,1), graphC(:,2), '.')
    axis('square',[0 1 0 1])

    %   imposing lines of weight vectors
    line([0 cos(pi/6)], [0 sin(pi/6)])
    line([0 cos(pi/4)], [0 sin(pi/4)])
```

```
            line([0 cos(pi/3)], [0 sin(pi/3)])

        %   jeff gray
        %   jhg7nm
        %   02.15.2016
        %   lab4
        %   file: catMod.m
        %   desc: category supervised synaptic modification

        %   parameter: 3xN input matrix of N vector inputs
        %   output: 2x1 synaptic weight vector
        function [synW] = catMod(inputMat)
            %   error checking
            if (size(inputMat, 1) ~= 3)
                disp('ERROR: input is not of size 3xN!')
            end

            %   forward declarations
            epsilon = 0.2;
            synW = ones(2,size(inputMat, 2));
            synW(:, 1) = 0;

            %   perform a moving average on inputs
            %   given a state of category activation
            for i = 1 : size(inputMat, 2)
                if (inputMat(1,i) == 1)
                    %   update first attribute
                    synW(1, i+1) = synW(1, i) + epsilon ...
                    * (inputMat(2, i) - synW(1, i));
                    %   update second attribute
                    synW(2, i+1) = synW(2, i) + epsilon ...
                    * (inputMat(3, i) - synW(2, i));
                end
            end

%   jeff gray
%   jhg7nm
%   02.15.2016
%   lab4
%   file: classifyVec.m
%   desc: declare category given inputs

%   parameter: 3xN input matrix of N vector inputs
%   output: 2x1 synaptic weight vector
function [category] = classifyVec(inputVec, thresh)
    %   error checking
    if (size(inputVec, 1) ~= 2)
        disp('ERROR: input is not of size 2xN!')
    end

    %   forward declarations
    weight1 = [cos(pi/6), sin(pi/6)]; % category 1
    weight2 = [cos(pi/4), sin(pi/4)]; % category 2
    weight3 = [cos(pi/3), sin(pi/3)]; % category 3
    thresh = thresh;

    stateA = dot(inputVec, weight1); % > cos(thresh);
```

```
    stateB = dot(inputVec, weight2); % > cos(thresh);
    stateC = dot(inputVec, weight3); % > cos(thresh);
%    output = [stateA, stateB, stateC]';


    disp("input category: ")
    if (stateA > stateB && stateA > stateC ...
        && dot(inputVec, weight1) > cos(thresh))
        category = 1;
    elseif (stateB > stateA && stateB > stateC ...
        && dot(inputVec, weight2) > cos(thresh))
        category = 2;
    elseif (stateC > stateA && stateC > stateB ...
        && dot(inputVec, weight3) > cos(thresh))
        category = 3;
    else %  if it doesn't fit in any
        category = 0;
    end


%   jeff gray
%   jhg7nm
%   02.15.2016
%   lab4
%   file: problem 4.1.2

function [output] =  problemTwo
    clc;
    clf;
    clear;
    rng(9711963);

    %   matrix centered at pi/3
    c = rand(100, 1) * pi/12 + pi/3 - pi/24;
    matC = ones(3, 100);
    for i = 1 : 100
        matC(1, i) = 1;
        matC(2, i) = cos(c(i,1));
        matC(3, i) = sin(c(i,1));
    end

    %   initializing function at arbitrary coordinates
    graphA = catModStart(matC, 0, 0)';
    graphB = catModStart(matC, 1, 0)';
    graphC = catModStart(matC, 0, 1)';
    graphD = catModStart(matC, 1, 1)';

    %   plotting resultant convergences
    plot(graphA(:,1), graphA(:,2))
    hold on
    plot(graphB(:,1), graphB(:,2))
    hold on
    plot(graphC(:,1), graphC(:,2))
    hold on
    plot(graphD(:,1), graphD(:,2))
    axis('square',[0 1 0 1])
```

```
%   jeff gray
%   jhg7nm
%   02.15.2016
%   lab4
%   file: problem 4.1.3

function problemThree
    clc;
    clf;
    clear;
    rng(9711963);

    a = rand(50, 1) * pi/12 + pi/6 - pi/24;
    matA = ones(3, 50);
    for i = 1 : 50
        matA(1, i) = 1;
        matA(2, i) = cos(a(i,1));
        matA(3, i) = sin(a(i,1));
    end

    b = rand(50, 1) * pi/12 + pi/3 - pi/24;
    matB = ones(3, 50);
    for i = 1 : 50
        matB(1, i) = 1;
        matB(2, i) = cos(b(i,1));
        matB(3, i) = sin(b(i,1));
    end

    %   adding and shuffling both sets
    mat = [matA, matB];
    ix = randperm(100);
    shuf = mat(:, ix)

    graphA = catModStart(shuf, 0, 0)';

    %   plotting resultant convergences
    plot(graphA(:,1), graphA(:,2))
    axis('square')
    hold on
    plot(matA(2,:), matA(3,:),'.')
    plot(matB(2,:), matB(3,:),'.')
```