Jeff Gray
02.02.2016
NESC 5330

Homework 2

Note: I discussed methods of dealing with problem 2.7.2 with classmate Berk Ekmekci after being unable to execute the problems using only radians. My result took more time because I couldn't optimize out the next step.

problem 2.7.2

The following data points represent sets that correspond with weight1 of π/6 radians, weight2 of π/4 radians, and weight3 of π/3 radians, respectively. As specified by the problem, no data points overlap into a separate region.

set 1 coordinates		set 2 coordinates		set 3 coordinates	
x1	y1	x2	y2	x3	y3
0.8986279647	0.438711501	0.6983511605	0.715755305	0.3996965614	0.9166475107
0.8846023127	0.466346168	0.7437889945	0.6684144909	0.4362413591	0.8998296931
0.8209707645	0.5709702302	0.6603897995	0.7509229739	0.5029916708	0.8642912583
0.9223288114	0.386405957	0.7622807087	0.647246569	0.5144712803	0.8575076103
0.9195106235	0.3930651514	0.6716994946	0.7408237233	0.5120165356	0.8589755918
0.9060505597	0.4231694497	0.7632023894	0.6461595103	0.5432441713	0.8395747557
0.8458513671	0.533418658	0.7310290333	0.6823463581	0.4980696369	0.8671370346
0.8341185246	0.551585249	0.6834740997	0.7299747633	0.4975558331	0.8674319529
0.8460425633	0.5331153544	0.6533755295	0.7570339605	0.4263411654	0.9045624416
0.8723778204	0.4888322192	0.7802460984	0.625472642	0.4317320192	0.9020019199

The advantage to setting the synaptic weight vector in the center of each category is that the threshold value in radians appears more relevant when plotting the data. This enables the range calculations to simply be the radian value of the weight vector +/- the threshold value. Doing this allows for a more efficient calculation and analytical process with respect to visualizing the data.

To optimize weight locations on the 2d hypersphere for n McCulloch-Pitts neurons, the weight vectors may be easily normalized to π/2/n+1 separate regions. This allows for the most even spacing in the final output graph and allows each individual neuron to accommodate any amount of potential inputs. This process results in equal arc-lengths for each weight vector and thus an even distribution.

problem 2.7.3

After setting up both MP neurons and assigning weight vector radians of π/6 and π/3 that exist symmetric to y=x, I began by detecting the errors associated with my initial threshold of π/12.

threshold	**type I error %**	**type II error %**
π/12	5	0
π/8	3	55

As demonstrated by the table above, the initial threshold displayed a mere 5% total error; however, after increasing the threshold to π/8, the type I error decreased by 2% while the type II error increased to a massive 55%.

By inspection, the average error percent positively correlates with threshold size in radians. When these points are not uniformly concentrated, I anticipate the average error to increase in conjunction with the added chaos to the overall plot. In the real world, the input concentrations exist undoubtedly as chaotic with significantly less order than these datasets mentioned so far in the lab. In conclusion, the precision of a point in space increases with a decrease in threshold.

problem 2.8.2

As specified by 2.8.1, the two input sets were calculated by calling the createinputs.m file:

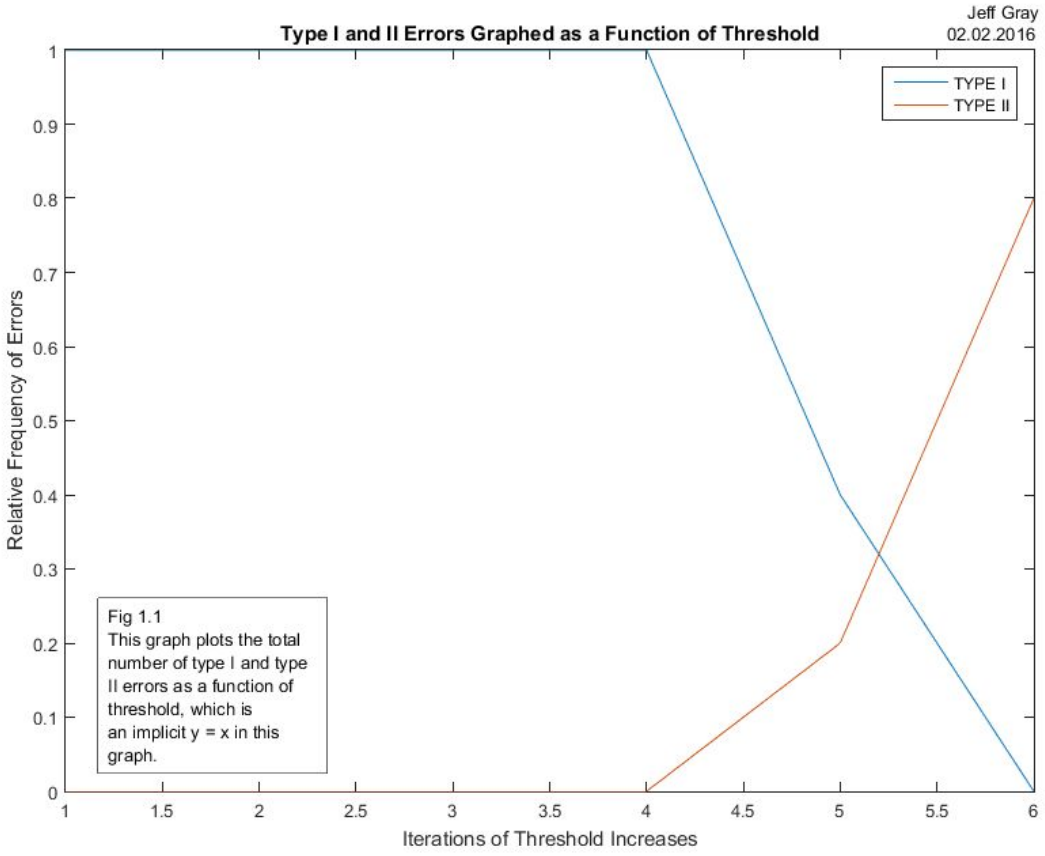
```
set1 = createinputs(5, 1, 5, pi/3, 0)
set2 = createinputs(5, 1, 5, pi/3, pi/6)
```

Afterwards, I utilized checkThresh.m in order to easily group together my outputs for analysis. As visible in the attached code, each iteration of the for loop outputs the number of iterations from the initial threshold. I specified quite a large range for these values: from π/24 to π/2. I've attached some sample output from the function to clarify an answer to the question.

iteration 3	iteration 4	iteration 5	iteration 6
out =	out =	out =	out =
0 0	0 0	0 0	1 1
0 0	0 0	0 0	1 1
0 0	0 0	0 1	1 1
0 0	0 0	0 1	0 1
0 0	0 0	1 1	1 1

As visible in the table above, the thresholds during iteration 3 and 4 were too small to account for any neuronal firing. The next increment of π/12 as well as iteration 6 reveal an overlap among the data sets.

The primary importance of this finding is that increasing threshold decreases type I error and increases type II error. This becomes evident with the increasing amount of double-fires evident as the threshold climbs while it iterates. The type I errors evident from the start eventually wane away as the threshold grows, allowing the values to be encapsulated in their range.



This previous graph serves to answer sections 3 and 4 of problem 2.8.2 by including all necessary data. The significant factor from this graph is that the optimal threshold rests slightly over π/3, represented here between iterations 5 and 6. The specific values are available in the code section, but this intersection represents the optimum threshold for the specified problem. This results in the most efficient minimization of error, because any point beyond this would sacrifice the problem by adding more type I errors, which are more dangerous. The reason type I errors are more dangerous is that they mean the base algorithm failed to encapsulate the data point rather than just a failure to set bounds.

CODE

note: ALL code visible on my github /jeffreygray/nesc5330

problem 2.7.2

```
%     code used to see active neuron firings and verify data points
inputs = rand(10,1) * scale + shift %    where user defines scale …
and shift to assign points onto unit circle
inputmatrix = [cos(inputs) sin(inputs)]
weightmatrix = [cos(weight) sin(weight)] %     where user defines ...
radian of weight vector
innerProduct = inputmatrix * weightmatrix'
innerProduct > cos(threshold) %    where user defines threshold
```

problem 2.7.3

```
%     create inputs
inputs = rand(90, 1)* pi/8 + pi/4 - pi/16
inputs1 = rand(5, 1)* .2945
inputs2 = rand(5, 1)* .2945 + .9817
allinputs = [inputs1; inputs; inputs2]
inputmatrix = [cos(allinputs) sin(allinputs)]

%     assign values and calculate inner products
weight1 = [cos(pi/6) sin(pi/6)]
weight2 = [cos(pi/3) sin(pi/3)]
ip1 = inputmatrix * weight1';
ip2 = inputmatrix * weight2';
```

problem 2.8.2

createinputs.m

```
%   jeff gray
%   jhg7nm
%   02.2.2016
%   nesc5330
%   lab1
%   file: 'createinputs.m'

%   description: creates input matrix of given dimension with random
values
```

```
function [inputs] = createinputs(row, col, numInputs, range, offset)
%   including numInputs despite its redundance

%   error detect
        if (row*col ~= numInputs)
            disp('ERROR: your input parameters do not agree! check
them.')
        end

        inputs = rand(row, col) * range + offset;
```

checkThresh.m

```
function checkThresh(weight, set1, set2)

    thresh = [pi/24 : pi/12 : pi/2]

    ip1 = set1 * weight;
    ip2 = set2 * weight;
    for i = 1 : length(thresh)
        disp(i)
        a = ip1 > cos(thresh(i));
        b = ip2 > cos(thresh(i));
        out = [a b]
        disp('----------------------------')
    end
```

thresholdOne.m % for plot

```
function thresholdOne(err1, err2)
figure1 = figure;

% Create axes
axes1 = axes('Parent',figure1);
box(axes1,'on');
hold(axes1,'on');

% Create multiple lines using matrix input to plot
plot1 = plot(err1);
hold
plot2 = plot(err2);
set(plot1,'DisplayName','TYPE I');
set(plot2,'DisplayName','TYPE II');
```

```
% Create text
text('Parent',axes1,'String','Jeff Gray',...
    'Position',[5.65918653576438 1.05321100917431 0]);

% Create text
text('Parent',axes1,'String','02.02.2016',...
    'Position',[5.55399719495091 1.02385321100917 0]);

% Create xlabel
xlabel({'Iterations of Threshold Increases'},'FontSize',11);

% Create ylabel
ylabel({'Relative Frequency of Errors'},'FontSize',11);

% Create title
title({'Type I and II Errors Graphed as a Function of Threshold'},...
    'FontSize',11);

% Create legend
legend(axes1,'show');
```

extra code used for lab completion:

```
%   02.01.2016
%   nesc5330
%   lab1
%   file: 'spikegen.m'

%   description: generates neuron output!

function [neuron_output] = getOutputs(innerProd, thresh)
%   threshold and excitation must be scalars

    neuron_output = innerProd > cos(thresh);

%   error detection
%    if length(excitation) > 1
%        disp('ERROR: spikegen.m takes a scalar not a vector or
matrix')
%    end

%    if excitation < threshold
```

```
%         neuron_output = 0;
%     else
%         neuron_output = 1;
%     end

%   jeff gray
%   jhg7nm
%   02.01.2016
%   nesc5330
%   lab1
%   file: 'postexcite.m'

%   description: calculates dendritic excitation

function [post_excitation] = postexcite(input_vector, weight)
    %   check for errors
%     checkvecs(input_vector, weight_vector, 'postexcite');
%     checkfor_col_vector(input_vector, length(weight_vector),
'postexcite');

    %   calculate excitation
    post_excitation = input_vector * weight;
%     post_excitation = dot(input_vector, weight_vector);
```