# Introduction to Neural Networks
U. Minn. Psy 5038

Problem Set 3

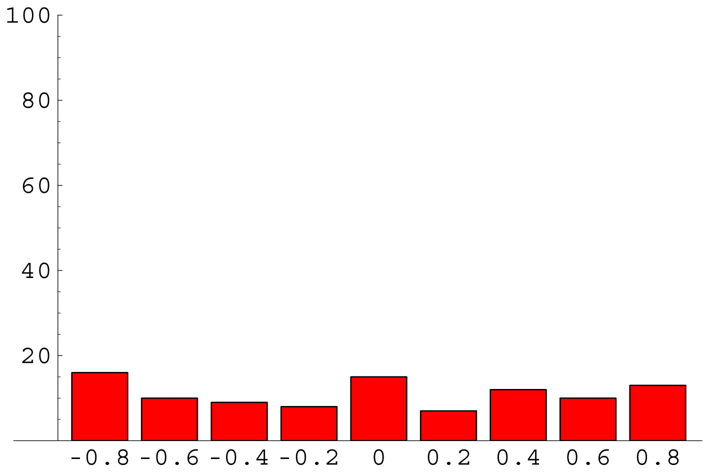
## Exercise 1 - Orthogonality of large random vectors

The assumption of orthogonality for the input patterns for the linear associator would seem to make it useless as a memory advice for arbitrary patterns. However, if the dimensionality of the input space is large, the odds are pretty good that the cosine of the angle between any two random vectors is close to zero. Make three histograms showing the distributions of the cosines of random vectors for dimensions 10, 50 , and 250. The histograms should put the cosines in bins from -0.9 to 0.9 in steps of 0.2. Sample the cosines one hundred times for each histogram.

If you wish, you can use the function **BinCounts[]** in **DataManipulation.m** and **BarChart[]** in **Graphics.m**. Here is an example to show the histogram for a uniform distribution between -0.9 and 0.9:

```
<<Statistics`DataManipulation`
<<Graphics`Graphics`
```

```
data[size_] := BinCounts[
                Table[1.8 Random[] - .9,{100}],
                {-.9,.9,0.2}];
BarChart[data[10], BarLabels->{-0.8, -0.6, -0.4,
                -0.2, 0, 0.2, 0.4, 0.6, 0.8},
                PlotRange->{0,100}];
```



```
- Graphics -
```

## Exercise 2 - Perceptron learning

Write a program that uses a Perceptron-style threshold logic unit that learns to classify two-dimensional vectors into "a" or "b" types. The unit should have three inputs: {1,x,y}, where x and y are the coordinates of the data to be classified. The first component, 1, is the standard "trick" used to incorporate the threshold into the weight vector. So three weights will have to be learned: {w1,w2,w3}. It may help to know something about Conditionals in *Mathematica*.

**Your report should have the following six sections. Some are done for you, but Sections 3, 5 and 6 require you to generate code.**

### ■ 1. Generation of synthetic classification data.

Generate 50 random points in the unit square, {{0,1},{0,1}} such that for the "a" type points,

x+y>0.6 and for the "b" points, x+y<0.4. Each pair of points should have its corresponding label, a or b.

```
stuff = {};size = 50;
While[Length[stuff]<size,
        Which[(x= .5 Random[]) + (y= .5 Random[]) < .4,
            stuff=Append[stuff,{"b",1,x,y}],
            x + y > .6, stuff=Append[stuff,{"a",1,x,y}]
            ]];
```

## ■ 2. Define threshold function

Define a function that is -1 for x <0 and +1 for x>= 0. Here we use the conditional **If[]**.

```
threshold[x_] := If[x<0,-1,1];
```

## ■ 3. Perceptron learning algorithm

Write a program that will run through your training pairs. Start off with a weight vector of: {-.3, -.05, 0.5}. If a point is classified correctly (e.g. as an "a" type), do nothing to the weights. If the point is actually an "a" type, but is incorrectly classified, increment the weights in some proportion (e.g. c = 0.1) of the point vector. If a "b" point is incorrectly classified, decrement the weight vector in proportion (e.g. c = - 0.1) to the values of coordinates of the training point.

Make a list to record the changes in the weight vector. Note that the three components of the weight vector determine the intercept and the slope of the line separating the "a" and "b" points.

Note that you may have to iterate through the list of training pairs more than once to obtain convergence--remember convergence is guaranteed for linearly separable data sets.

## ■ 4. Table of classifications

Make a list {{c1,out1},c2,out2},,,,} where ci is the correct classification, and outi is the output of your threshold logic unit with weight w or w0.

```
Table[{stuff[[i,1]],threshold[Take[stuff[[i]],-3].w]},
        {i,1,Length[stuff]}]
```

```
{{b, -1}, {a, 1}, {b, -1}, {b, -1}, {b, -1}, {a, 1}, {a, 1},
  {a, 1}, {a, 1}, {a, 1}, {a, 1}, {a, 1}, {b, -1}, {b, -1},
  {b, -1}, {a, 1}, {a, 1}, {b, -1}, {a, 1}, {b, -1}, {a, 1},
  {a, 1}, {b, -1}, {a, 1}, {a, 1}, {b, -1}, {b, -1}, {a, 1},
  {a, 1}, {b, -1}, {b, -1}, {b, -1}, {b, -1}, {b, -1},
  {a, 1}, {a, 1}, {a, 1}, {b, -1}, {a, 1}, {b, -1}}
```

## ■ 5. Proportion of correction classifications - before and after

Write a procedure (or function) that will give the proportion of correct classifications before and after the weight vector has been trained.

## ■ 6. Plots of discriminant line

Make a series of plots showing how the weights evolve through the learning phase. Show the input data to be classified on the same plot. Don't show all of the iterations (there would be too many and you might run out of space), but show a subset of about 10. By grouping your plots, you can use animate selected graphics (in the **Graph** menu) to animate them and watch your discriminant line move around until it has finally settled into a region that works.

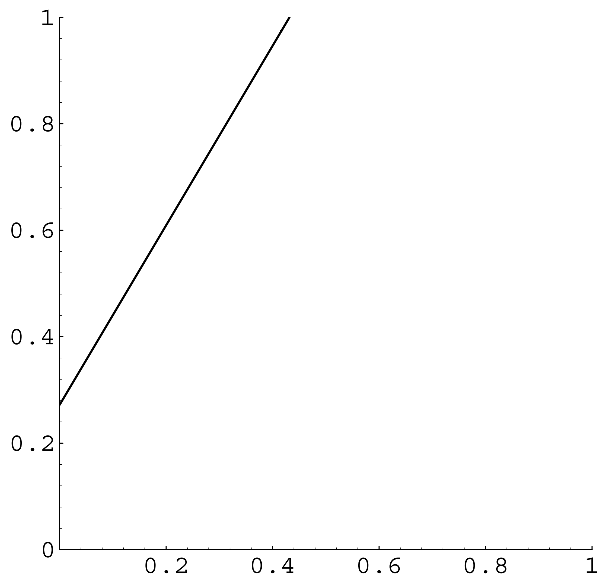
```
 w={a,b,c};

Plot[(-c/b) x - (a/b), {x,0,1},PlotRange->{{0,1},{0,1}},

   AspectRatio->1];
```

Here is an example:

```
For[i=1,i<Length[wlist],i=i+8,
    a = wlist[[i,1]]; b = wlist[[i,2]]; c = wlist[[i,3]];
    Plot[(-c/b) x - (a/b), {x,0,1},PlotRange->{{0,1},{0,1}},
    AspectRatio->1];
]
```


1
0.8
0.6
0.4
0.2
0
0.2
0.4
0.6
0.8
1