Introduction to Neural Networks
U. Minn. Psy 5038

Problem Set 3

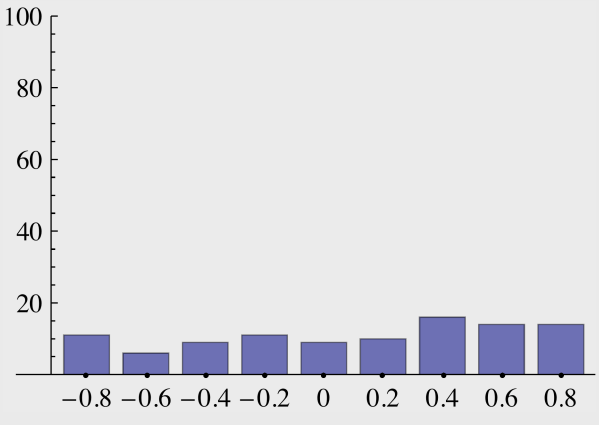
Exercise 1 - Orthogonality of large random vectors

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[]** and **BarChart[]**, which needs the package **BarCharts** to be loaded (as below). Here is an example to show the histogram for a uniform distribution between -0.9 and 0.9:

```
Needs["BarCharts`"]
```

```
data[size_] := BinCounts[Table[1.8` RandomReal[] - 0.9`, {100}],
  {-0.9`, 0.9`, 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}]
```



Exercise 2 - Regression and Widrow-Hoff learning

Make a function: **rline[slope_,intercept_]** to generate pairs of random numbers {x,y} where x ranges between 0 and 10, and whose y coordinate is a straight line with slope, **slope_** and intercept, **intercept_** but perturbed by additive uniform random noise over the range -2 to 2.

Generate a data set from **rline** with 200 samples with slope 11 and intercept 0.

Use the function **Fit[]** to find the slope and intercept of this data set. Here is an example of how **Fit[]** works:

```
Fit[{{1,3},{2,5},{3,7.1}}, {1,x}, x]
```

```
0.933333 + 2.05 x
```

If you wanted to fit a line through the origin, you would write: **Fit[{{1,3},{2,5},{3,7.1}}, {x}, x**], and a quadratic fit as: **Fit[{{1,3},{2,5},{3,7.1}}, {1,x,x^2}, x]**.

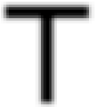
Now implement a Widrow-Hoff algorithm to find the slope of the data. Assume the data goes through the origin, and initialize the slope to zero for the first iteration. Use **ListPlot** to show the values of the slope as a function of the iterations from 1 to 200. What is a suitable range for the learning constant?

Exercise 3 - The biggest eigenvector of an autoassociative memory

An early model of recall named the "brain-state-in-a-box" model or BSB model (James Anderson) showed that a generic neuron model that feeds its output back to its input will under certain conditions converge to the largest eigenvector of the association matrix. Assume memories are formed by the simple autossociation using the outer product rule.

```
Pmatrix = {
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 1, 1, 1, 1, 0, 0, 0, 0, 0},
  {0, 1, 0, 0, 0, 1, 0, 0, 0, 0},
  {0, 1, 0, 0, 0, 1, 0, 0, 0, 0},
  {0, 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, 0}};
```

```
Imatrix = {
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};
```



```
Tmatrix = {
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
  {0, 0, 0, 0, 1, 1, 1, 1, 1, 0},
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};
```

```
Tv = Normalize[Flatten[N[Tmatrix]]];
Iv = Normalize[Flatten[N[Imatrix]]];
Pv = Normalize[Flatten[N[Pmatrix]]];
```

Calculate the autoassociative weight matrix **W** which has "seen" the **T** 10 times, the **I** 3 times, and **P** only once.

Find the eigenvector of the weight matrix **W** with the biggest eigenvalue. Use the **Partition** function to represent the eigenvector as a 10x10 picture, and then make an **ArrayPlot[]** of this eigenvector.

Now use **Nest[]** to recursively apply **W** 8 times as a dot product to a random *initial* starting vector **x=Table[RandomReal[{0, 1}], {100}]**;. Note that **x** is fixed, so you want to use Nest to calculate:

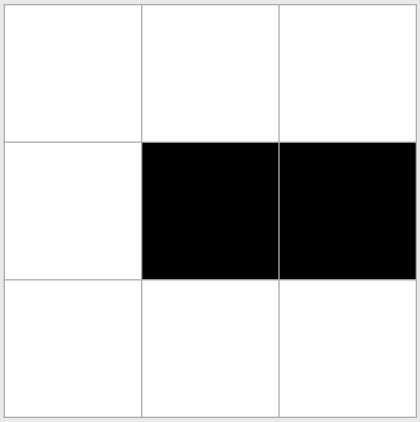
W[W[W[W[W[W[W[W[x]]]]]]]]. Again use the **Partition[]** function to represent the result as a 10x10 picture, and then make an **ArrayPlot[]**.

Exercise 4- Use backprop to classify letters independent of orientation

Define two classes of patterns, T's and C's. Each class has four members for rotations of 0,90,180, and 270 degrees. Train a non-linear feedforward net with one layer of hidden units to correctly classify the eight patterns as T or C. To get you started, here are exemplars for T and C on a 3x3 grid:

```
T1 = {{0.9,0.9,0.9},{0.1,0.9,0.1},{0.1,0.9,0.1}};
C1 = {{0.9,0.9,0.9},{0.9,0.1,0.1},{0.9,0.9,0.9}};
```

```
ArrayPlot[C1, Mesh → True, ColorFunction → GrayLevel, AspectRatio → 1,
 Frame → False]
```



Fix the learning constant at 1. What is the minimum numbers of hidden units you can find that will still converge in under 600 iterations?

For this number of hidden units, what is the maximum value of the learning constant (eta) that still allows convergence? You can use Backpropagation.m for this exercise.