Introduction to Neural Networks
U. Minn. Psy 5038
Spring, 1999
Problem Set 4

Exercise 1 - 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 it 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 2 - The biggest eigenvector of an autoassociative memory

In Anderson's BSB model of memory, memories are can be formed by the simple autossociation.

```
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}};
```

```
General::spell:
   Possible spelling error: new symbol name "Pmatrix"
     is similar to existing symbols {Imatrix, Tmatrix}.
```

```
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}};
```

```
General::spell1:
   Possible spelling error: new symbol name "Imatrix"
     is similar to existing symbol "Pmatrix".

General::spell1: Possible spelling error: new symbol name "Imatrix"
     is similar to existing symbol "Pmatrix".

General::spell1:
   Possible spelling error: new symbol name "Imatrix"
     is similar to existing symbol "Pmatrix".

General::spell1:
   Possible spelling error: new symbol name "Imatrix"
     is similar to existing symbol "Pmatrix".

General::spell1:
   Possible spelling error: new symbol name "Imatrix"
     is similar to existing symbol "Pmatrix".
```

```
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, 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}};
```

```
General::spell:
   Possible spelling error: new symbol name "Tmatrix"
     is similar to existing symbols {Imatrix, Pmatrix}.
```

```
normalize[x_] := N[x/Sqrt[x.x]];
Tv = normalize[Flatten[Tmatrix]];
Iv = normalize[Flatten[Imatrix]];
Pv = normalize[Flatten[Pmatrix]];
```

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

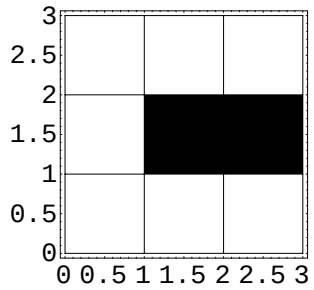
Find the eigenvector of the weight matrix with the biggest eigenvalue. Make a **ListDensityPlot** of this eigenvector.

Exercise 3- 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}};
```

```
ListDensityPlot[C1];
```



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?

Exercise 4 - Oja's rule for weight growth

Recall that there is nothing in the outer product version of the Hebb learning rule to prevent the weights from growing without bound. Assume a linear model of a neuron with n inputs:

y = w1*x1 + w2*x2 +...+wn*xn.

Show that Oja's modification of the Hebb rule tends to normalize the weights according to the following rule:

Sum[w1^2+w2^2+...+wn^2] = 1

Your proof needn't use *Mathematica*.

Notes on continuous response Hopfield network (from Lecture 16)

Definitions and functions that you will find useful for the exercises below in this problem set are reproduced from Lecture 16 in this section.

■ Definitions

We will let the resistances and capacitances all be one, and the current input I_i be zero. Define the sigmoid function, **g[]** and its inverse, **inverseg[]**:

```
a1 := (2/Pi); b1 := Pi 1.4 / 2;
g[x_] := N[a1 ArcTan[b1 x]];
inverseg[x_] := N[(1/b1) Tan[x/a1]];
```

■ Initialization of starting values

The initialization section sets the starting output values of the two neurons

V = {0.2, -0.5}, and the internal values **u = inverseg[V]**, the step size, **dt=0.3**, and the 2x2 weight matrix, **Tm** such that the synaptic weights between neurons are both 1. The synaptic weight between each neuron and itself is zero.

```
dt = 0.3;
Tm = {{0,1},{1,0}};
V = {0.2,-0.5};
u = inverseg[V];
result = {};
```

■ Main Program

The following function computes the output of the ith neuron for the network with a list of neural values **uu**, and a weight matrix **Tm**.

```
Hopfield[uu_,i_] := uu[[i]] + dt ((Tm.g[uu])[[i]] - uu[[i]]);
```

Let's accumulate some results for a series of 80 iterations. Then we will plot the pairs of activities of the two neurons over the 80 iterations.

```
result = Table[{k=Random[Integer,{1,2}],u[[k ]] = Hopfield[u,k],u}, {80}];
```

```
result = Transpose[result][[3]];
gresults = ListPlot[g[result],
    PlotJoined->False,AxesOrigin->{0,0},
    PlotRange->{{-1,1},{-1,1}},
    Frame->True, AspectRatio->1,Ticks->None];
```

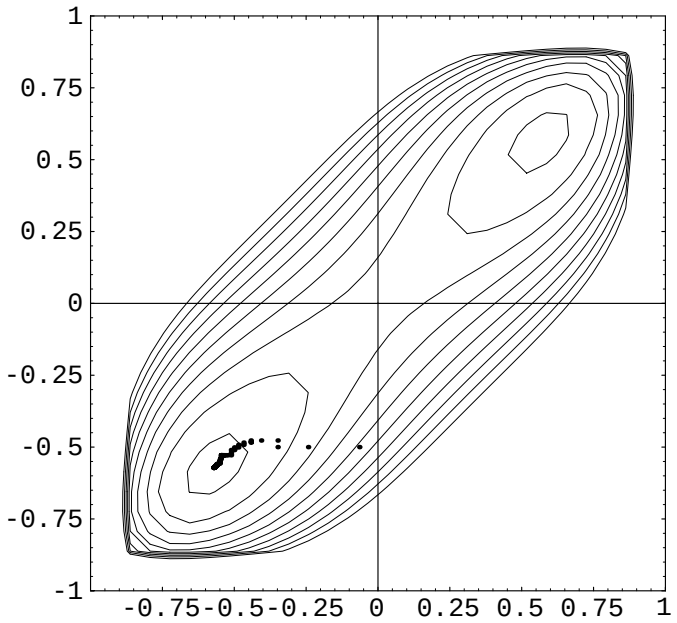
■ Energy landscape

Here is a contour plot of the energy landscape.

```
inig[x_] := -N[(a1*Log[Cos[x/a1]])/b1];
energy[Vv_] := -0.5 (Tm.Vv).Vv +
    Sum[inig[Vv][[i]], {i,Length[Vv]}];
```

```
gcontour = ContourPlot[energy[{x,y}],{x,-1,1},{y,-1,1},
    AxesOrigin->{0,0}, ContourShading->False,
    PlotRange->{-.1,.8}, Contours->32, PlotPoints->30];
```

```
Show[gresults, gcontour];
```



Exercise 5 - Hopfield network

Do the above simulation (in "Notes on continuous...") of a 2-neuron continous response Hopfield net with a starting value of **V** that will allow the state vector to be drawn to the other attractor basin. Once again, plot the trajectory superimposed on the energy contour plot.

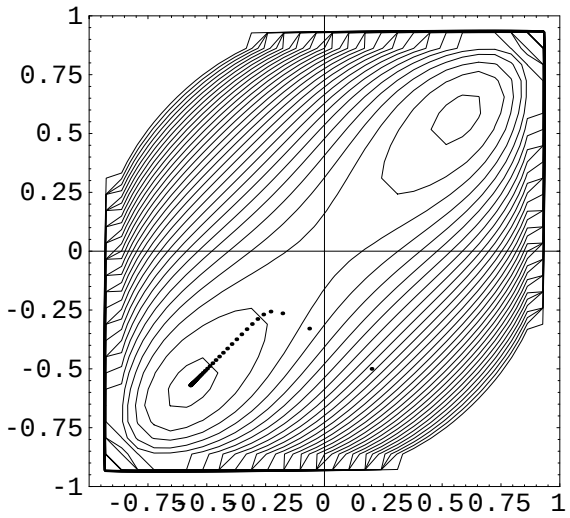
Exercise 6 - Synchronous updating for Hopfield network

Define a function **syncHopfield[uu_]** to do deterministic synchronous updating rather than the asynchronous updating rule specified in the above Hopfield network.Use **results = NestList[syncHopfield,*,*]** with 40 iterations to list the state space trajectory. Plot the results as shown below.

Use the same starting values used in Lecture 16 (see also above).

Now use asynchronous iterations (as in the above insert from Lecture 16) to compute 20 trajectories all from the same starting point. Compute the average trajectory, and plot both this average trajectory and the deterministic synchronous trajectory together on the same contour plot to compare the descent with the asynchronous case.

Which one appears to be closest to a steepest descent?


1
0.75
0.5
0.25
0
-0.25
-0.5
-0.75
-1
-0.75 -0.5 -0.25 0 0.25 0.5 0.75 1