

### Project 3: Neural Networks

#### Program description

Write a program to implement the delta learning rule for a two-layer network. Use the following simple case:

- 1 Initially we take the the activations  $a_i$  to be either 0 or 1.
- 2 The activation function is either the step-function (eqn. (5) of the notes, with the  $f(0) = 1$ ) or the sigmoidal function (eqn. (6) of the notes).
- 3 The weights are initially set at random in the interval  $(-0.5, 0.5)$ .
- 4 The thresholds  $\theta_i$  are zero.

Your program should have the following features:

- 1 Input teaching sets from a data file, thus saving much re-typing.
  - 2 Allow for different values of the learning rate parameter  $\eta$ .
  - 3 The program should stop (with an informative message) if convergence is not attained within a certain number of epochs.
  - 4 Allow for flexible printout of the weight matrix; e.g. every step, every epoch, every specified number of epochs, or only when convergence is attained.
  - 5 After convergence, the program should allow the user to input further patterns and determine the output patterns, with no further learning taking place.
1. Run a 5 neuron network with 3 input and 2 output neurons, and the patterns (input followed by output):

1 1 1	1 0
0 1 0	0 1
0 0 1	1 1

Using  $\eta = 1$ , and a step-function activation function do runs with full, partial and minimum printout. Then with minimum printout, do runs with  $\eta = 10, 1$  and  $0.1$  and observe the differences in the number of cycles taken to convergence and in the final weights found. Describe your findings. Now try the sigmoidal activation function (see question 5 for a minor modification that is required to the learning rule). Compare your results with those for the step-function and explain the differences.

2. One of the uses of these networks is as *error-correction devices*; a network trained on a number of patterns may still give the trained output even if the input is slightly distorted. Demonstrate this by training a 8-input to 4-output network on the patterns:

1 1 1 1 0 0 0 0	1 1 0 0
0 0 0 0 1 1 1 1	0 0 1 1
1 1 0 0 0 0 1 1	1 0 0 1

(Use  $\eta = 1$ .) Then test the response of the system to input patterns which differ from the above in 1 place. (You should find that in almost all cases the output for the undistorted pattern is still obtained.) Repeat with changes in 2 places.

3. Apply the network to the following cases, using the step-function activation function.

First consider:

0 0	0
0 1	1
1 0	1

Show, by considering the updating algorithm (eqns. (2), (3) and (5) of the notes) that failure is inevitable for this example, so long as  $f(0) = 1$  for the step-function. However, if  $f(0) = 0$  then there is no failure!

Demonstrate, using the program, that the same network cannot deal with the exclusive or problem:

0 0	0
0 1	1
1 0	1
1 1	0

Now use the network on the following generalization and explain whether you would expect this to work:

0 0 0	0
0 1 0	1
1 0 0	1
1 1 1	1

Finally, demonstrate that the network cannot successfully deal with the following symmetry recognition problem, where a symmetric input gives an output 1, and otherwise the output is 0:

1 1 1	1
0 0 0	1
1 0 1	1
0 1 0	1
0 1 1	0
1 1 0	0
0 0 1	0
1 0 0	0

Try to find the largest subset of these conditions that the network can handle.

4. Make a new version of your code that includes a hidden layer and uses the back-propagation algorithm to train the network. Test using some of the examples covered in the previous exercises, particularly those which could not be successfully dealt with using a two-layer network. Experiment with the number of hidden nodes and determine how this affects the convergence.

Updating of weights for each node  $i$  in the output layer is as for the two-layer network, i.e.

$$\Delta W_{ij} = \eta \delta_i x_{ij}$$

where  $x_{ij}$  are the inputs to node  $i$  (generalisation of  $a_i$ , equation (3) of notes) and  $\delta_i = (t_i - f(h_i))f'(h_i)$ , with  $f(x)$  the activation function,  $t_i$  the target output for node  $i$  and  $h_i$  defined in equation (2) of the notes.

The weights for each node  $i$  in the hidden layer are updated according to

$$\Delta W_{ij} = \eta f'(h_i) \delta_i x_{ij}$$

where  $\delta_i = \sum_k \delta_k W_{ki}$ , with the sum taken over the output layer.

5. The purpose of this exercise is to use the hidden-layer network to recognise a representation of some letters of the alphabet. For each of the (capital) letters A,B,C set up a 5 by 5 grid, with a suitable pattern of zeros and ones to represent each letter. Treat the 25 grid values as input, and use as output the values 1.0, 0.75 and 0.5 for A,B,C respectively. Train the network to distinguish these letters from say 5 or 6 grids of random entries, all of which should be assigned output 0. Use the sigmoidal activation function. Once you have trained the network, try to recognise the three letters in the presence of noise (generate the perturbations randomly). From your results estimate the maximum noise levels under which the network can still distinguish the letters.

6. This exercise applies the hidden-layer network to the problem of using underlying patterns in the betting on horse races to attempt to become rich (N.B. no responsibility will be taken for any bankruptcies that ensue).

You are provided with a file containing 136 lines of six entries called **horses.dat**. This file can be obtained from the website. After the preamble, each line of the file corresponds to a race. The last entry in each line tells you whether or not the favourite for the race won (1=win). The first 5 entries correspond to the (natural) logarithms of the totes. The first tote is the return to a person betting \$1 on the favourite, *if* the favourite wins. Note that the profit is then \$1 less than this. The second tote is the return to a person betting \$1 on the second favourite and so on. We are only told whether the favourite won or lost in the sixth entry; if the second favourite won, we have no way of telling this from the information given.

The task is to use the log totes as inputs to train the network, where the single target output is the sixth entry in each row. Use the sigmoidal map and the delta learning rule, but use the sum of errors as your estimate of the total error, as opposed to using the sum of the squared errors. Once the network has been trained, the output of the network can be regarded as a probability, say  $p_i$ , of winning a given race. Produce a histogram of the  $p_i$ , using e.g. the **MATLAB** command **hist**. Note that this is somewhat removed from the bi-modal 0-1 distribution that actually occurs.

Then use the following strategy to make your fortune. Train the network, using the first 50 races, say. For the next 86 races, bet \$1 on the favourite if your expected profit is greater than zero, i.e. if  $p_i \times$  the expected return (the exponential of the first log tote) is greater than one. Now work out your actual return (you know the result of the race) and see if you come out ahead. Compare this result with the simple strategy of betting \$1 on the favourite in each race. See if you do better or worse with other strategies, e.g. betting a greater amount if the expected return is higher and check how dependent your results are on the number of races used to train the network.

[Notes: (a) for this exercise use initial weights in the interval (0, 1);

(b) experiment with different values of the learning rate parameter  $\eta$ ;

(c) try getting the network to work with a small number of patterns first.

## Project report

This should include:

A brief overview of the methods used.

Discussion of each of the exercises, describing the methods used, and making reference to and explaining the results obtained.

General summary of the two-layer and hidden-layer neural network techniques, outlining their strengths and limitations.

A program listing.