

**Homework 4, posted Jan 31, 2008**  
**Due Feb 14, 2008**

**1. (25 points)**

Implement, in MATLAB, the simple, concrete BP network that is given as a sample exercise on pages 44-45 of Colin Fyfe's CF1/4 chapter The Multilayer Perceptron: backprop.

(The requirement in this problem is just a "hardwired" network. You may, of course, choose to write a code for a 2-layer BP with variable number of weights etc., and set the variables to reflect the concrete example.)

- Choose 1 for the slope parameter of the tanh(.) transfer function.
- Initialize the weights to small random numbers. (You can use Colin's numbers.)
- Set the learning parameter to 0.001
- Use on-line training (epoch size = 1).
- Set a stopping criterium, e.g., 'stop when a certain number of iterations has been performed or stop when the absolute value of the error or RMS error at the output is less than epsilon, whichever comes first', or 'stop when a certain number of iterations is reached provided the error is below epsilon'. Epsilon should be small, such as 0.05 .
- Train this network to learn the logical XOR function. Represent your inputs and outputs as follows:

Bias	x1	x2	output (= x1 XOR x2)
1	1	1	-1
1	1	0	1
1	0	1	1
1	0	0	-1

- Document your training process and the results in the following way:
  - a) Print the network parameters, including the initial weight values, stopping criteria, learn parameter, tanh slope.
  - b) For every  $m$  training steps tabulate the input and output (to see the progress of learning). Choose  $m$  reasonably, i.e., if it takes only a hundred or less steps for the learning to converge, print the input/output for every 10 steps. If it takes 10 or so steps, print for every step, etc. (This tabulated output is not typically done when monitoring a network's progress. We will only do it for this very simple case.)
  - c) Record performance indicators of the network for every  $m$  steps, and either tabulate or plot them. Indicators should be
    - The RMS error or absolute error at the output (as a function of epochs), for the training data
    - After training is finished, a plot of desired outputs vs actual outputs for all training samples.
  - d) Print the number of iterations that the network took to learn the XOR function.
  - e) Attach a copy of your MATLAB code.

**2. (40 points)**

Modify the network that you implemented in Problem 1.), to be a general 2-layer BP network with variable numbers of input, hidden and output PEs and weights. (If you made it general in 1. that saves you this step. The number of layers can still be hardwired here.) Extend it to be able to do batch learning (to sum up the squared errors for  $K$  training samples before one weight update is done). Train this network to learn the function  $f(x) = 1/x$  in  $(0.1, 1.0)$ , using the following configuration:

- 10 hidden PEs
- slope parameter for  $\tanh(\cdot) = 1$
- Generate 200 training data pairs  $(x, 1/x)$   $0.1 < x < 1.0$ . (You may generate these in a random order and store in a matrix from which you can then take them sequentially – not exactly a random picking scheme, but may be adequate for this problem)
- Generate 100 test data pairs, on which you will measure the network's ability to generalize, after every  $m$  iterations.
- Scale your input and output data according to the transfer function, i.e., to fall between  $-1$  and  $1$  (you can do this at the time of training/test data generation). Note that since the input training/test data (the  $x$  values) are already in the appropriate range since we work in  $(0.1, 1.0)$ , you only need to scale the output (training and test) data. Note also that the range of the transfer function does not have to be "filled", i.e., since your output data are all positive scaling them to fit into the  $(0, 1)$  range by dividing by the maximum value of the output data is fine.
- Use learning rate = 0.05.
- Set maximum number of iterations to 3000.
- Set epoch,  $K = 200$  (equal to the number of training data points).
- Document your training process and results the same way as in 1), with the exception of  $1/b$  (do not tabulate the input-output for each step). In addition, test your network on the test data set at the same times as you test for training accuracy, and plot the test errors too (best is to plot them in the same window); and plot the actual vs desired outputs for all points of the test data set too.

Tips: you may write a separate routine for testing the network, i.e., to run training and test data through the network with frozen weights and calculate errors, for the purpose of verification of performance.

Remember to scale back the output data into the original range before plotting against the original target function values.

If your network does not converge at all, you may try a different learning rate, but beware that the problem may lie elsewhere than in the learning rate. If you are getting reasonable results, save the experimentation with the learn rate for the next point of this HW.

**3. (25 points)** Experiment with various numbers of hidden PEs for the same problem as in 2. Test the generalization performance of the network at every  $m$  iterations, say  $m=1000$ . (If  $K = 200$ , one epoch consists of 200 iterations, so  $m=1000$  is reached in 5 epochs.) If the generalization capability seems to increase with the number of iterations, you may allow more

steps. Otherwise you may want to revert to the number of iterations where the test results were best. (For this reason you may want to save your network at every  $m$  iterations.) Similarly, you can experiment with other parameters, such as the learning rate or the number of training samples, to achieve better learning, and better prediction for the test data.

Document your best results (or in case of getting only poorer results than in 2., document one of those), and say why you think the network's performance increased or decreased as compared to 2.) (Here you do not have to attach a copy of the same MATLAB source code again.) The purpose of this exercise is to get a feel for the effect of changing the network parameters, not to have you do exhaustive work. What I am looking for is to see that you explored your network's capabilities.

**4. (30 points)** Train a new copy of the above network to classify the 'iris' data set. Use the version of the iris data that is already split into a training and a test data set (iris-train.txt and iris-test.txt). Document your best result as in 2) and 3), and in addition briefly report your experiments (e.g., what network configuration did you start with, did you need to change it in order to achieve satisfactory training results and test results, if so, what changes did you make, and how did the network's performance change as a result of that).

Again, you do not need to do hundreds of network training trials in order to get a little better performance out of your network. The goal is similar as in 3.

The iris data are in /home/erzsebet/ANNclass502 on hoss.ece.rice.edu.