

# Project 2

CS 4793 – Fall 2003  
Tom Bylander, Instructor

assigned October 23, 2003  
due November 25, 2003

This project is designed to be performed by groups from 1 to 4 people. A group of  $n$  students will implement  $n$  of the following tasks, which will be explained in more detail farther below:

1. Backpropagation neural networks with momentum and hypertangent activations. This is required for all groups.
2. Automatic scaling of datasets.
3. Selection of weights using a validation set.
4. 10-fold cross-validation testing.

As in Project 1, you should implement both an incremental learning and batch learning version of backpropagation. Add a “-b” option to the command line to select batch learning. Batch learning is implemented by calling `update_neuron` only once at the end of each epoch. The learning rate should be divided by the number of examples.

In addition, a group of  $n$  students will use  $n$  classification datasets and test the algorithms on these datasets. These can be the same datasets used in Project 1.

Unless you implement 10-fold cross-validation, testing will be performed using the holdout method. The holdout program from the initial source code will partition a dataset into a training set and a test set. An option on the command line will run the learned network on the test set. Perform the testing using 1000 and 10000 epochs on both incremental and batch learning. Also test different numbers of hidden neurons (I suggest 5 and 10) and different momentum values (I suggest 0.0 and 0.5). You will have to experiment to find a reasonable learning rate. Also include the `iris.data` dataset (available on the web site) in your testing.

## Tasks

This describes the learning algorithms listed above in more detail. As in class,  $\mathbf{x}$  is a vector of inputs and  $\mathbf{d}$  is a vector of desired outputs. Let  $X$  be the maximum length of any input vector. There will be as many neurons as outputs. For a given neuron,  $u$  is  $b + \sum_i w_i x_i$ ,  $o = a(u)$  is the output where  $a$  is the activation function, and  $d$  is the desired output. I will assume  $d$  is either +1 or -1. For datasets that use 1 and 0 as outputs, the 0 should be treated the same as a -1.

## Backpropagation Neural Networks

The number of hidden neurons is specified using a “-h” option on the command line. Similar to the perceptron and adaline algorithms, the number of output neurons should be equal to the number of outputs in the dataset.

The “len” field of a hidden neuron should be equal to the number of inputs. The “len” field of an output neuron should be equal to the number of hidden neurons. By default, use hypertangent activation for both the hidden and output neurons.

When the neural network is run on a vector of inputs, the hidden neurons should be run on the inputs, and the output neurons should be run on the outputs of the hidden neurons.

After the neural network has been run, then `backprop_neuron` should be applied to each output neuron using  $o - d$  as the second argument (the output of the neuron minus the desired output). This procedure will set the “dx” fields of the output neurons. The “dx” field of an output neuron is an array whose length is the number of hidden neurons.

Next, `backprop_neuron` should be applied to each hidden neuron. The second argument for the first hidden neuron should be the sum of the first “dx” value from each output neuron. Likewise, the second argument for the  $n$ th hidden neuron should be the sum of the  $n$ th “dx” value from each output neuron.

If incremental learning has been selected, then `update_neuron` should be called on each neuron. If batch learning has been selected, then `update_neuron` is not called until each example has been processed. For batch learning, the learning rate should be divided by the number of examples.

## Scaling of Datasets

The neural network FAQ suggests two ways of scaling the inputs. Read the section “Should I standardize the input variables?”. Implement both methods suggested there and allow the selection of a scaling method on the command line.

When scaling/standardization is performed, both the training set and test set need to be scaled. However, the parameters are calculated only using the training set.

## Validation Set Training

Implement an option “-v” that will select training using a validation set. In this case, the dataset should be randomly split into two datasets, a learning set and a validation set. For extra credit, this split should be stratified, i.e., if there are  $k$  examples of class  $c$ , then each half should have either  $\lfloor k/2 \rfloor$  or  $\lceil k/2 \rceil$  examples of class  $c$ .

For each epoch, the neural network is updated using the learning set. After the update, determine the loss of the neural network on the validation set. If this loss is the lowest so far, then the weights of the neural network need be copied. At the end, the weights of the neural network will be replaced with the weights that achieved the lowest error on the validation set.

## 10-Fold Cross-Validation

Use a “-c” option to select 10-fold cross-validation on the command line.

10-fold cross-validation is a common technique to evaluate learning algorithms on a dataset. In this method, a dataset is randomly partitioned into 10 subsets (called “folds”). If there are  $m$  examples, then each fold should have either  $\lfloor m/10 \rfloor$  or  $\lceil m/10 \rceil$  examples. For extra credit, ensure that the folds are stratified, i.e., if there are  $k$  examples of class  $c$ , then each fold should have either  $\lfloor k/10 \rfloor$  or  $\lceil k/10 \rceil$  examples of class  $c$ .

For each fold, the following processing is performed. For this iteration, the selected fold becomes the test set, and the other 9 folds are combined into the training set. Create a neural network and train it on the training set and test it on the test set. Sum the test loss over the iterations to obtain the estimated performance of the neural network.

## **Submission**

To submit your project, email it to [cs4793@ai.cs.utsa.edu](mailto:cs4793@ai.cs.utsa.edu). Your source code should be attached as a tar or zip file. Do not include object files or executable files in your submission. I should be able to use “make all” to compile and link your code. You should also include a report indicating what algorithms you implemented and displaying your test results. Describe any design decisions that you made and how you implemented them.