Artificial neural networks, advanced course, DD2433

Lab 1: Accelerated Back Propagation and Regularisation via Pruning

March 23, 2008

1 Objectives

The primary objective of this exercise is to give you some experience with the Neural Network Toolbox for Matlab. You will have to look in the on-line documentation to find out how to organise the data, use the different algorithms and to access the results.

Your task will be to design a multi-layer perceptron for a non-trivial classification task. In doing so, you should also get some experience with back-propagation and related methods. In particular, you should find out:

- How back-propagation learning time scales with problem size
- How much faster learning is with an accelerated method
- How automatic regularisation helps

2 Dataset

All initial experiments will be done using the "wine" dataset. This dataset consists of 178 samples of wine, each comprising 13 real valued attributes corresponding to the outcome of various chemical analyses. Each sample belongs to one of three classes, depending on where the grapes were growing. More detailed information about the dataset is available in the file `wine.names` in the course directory. The actual sample values are available in the file `wine.data` in a format which can be loaded directly into Matlab with the `load` command.

When loaded into Matlab, the data is represented as a 178×14 matrix where each row corresponds to one sample. The first column contains the classification; numbered 1, 2 and 3. The remaining columns contain the 13 attribute values. Note that the rows are ordered according to class so it is necessary to randomly permute the rows before separating the training and test sets.

We will regard this as a classification task. Use three output nodes; one for eash class. You need to transform the class number in the first column into a properly formatted target matrix.

3 Tasks

All the tasks will be done using a two-layered feedforward network. You will have to experiment with the parameters of the algorithms in order to obtain a network which performs well on both the training set and the testing set.

You will compare the generalization performance using different learning strategies and different numbers of hidden units. Design your Matlab script so that you get a scalar measure of the generalization performance as a result. The natural measure is to calculate the percentage of correctly classified samples in the test set.

3.1 Training with BackProp

Use the `newff`, `train` and `sim` functions from the NN toolbox to, respectively, build, train and evaluate the network. Use the built in documentation to find out how to organize the data and call these functions. Start by using the default parameters, but be prepared to adjust some parameters if necessary.

Try to find an optimal number of hidden units where generalisation performance is good.

3.2 Conjugate Gradient Method

The Conjugate Gradient Method is a numerical technique which automatically selects the optimal step length, separately in eash direction in the weight space. This means that you will have fewer parameters to set manually. In theory, the Conjugate Gradient Method should converge faster than the ordinary BackProp algorithm. Use the `trainscg` function from the NN toolbox to see how much speedup you get in this case.

Since the algorithm now runs faster, you can afford to make a systematic search to find the optimal number of hidden units. Plot the performance (on the test set) as a function of the number of hidden units.

3.3 Weight Pruning

Pruning is a way of automatically reducing the risk of generating unnecessarily complex networks. The easiest way to accomplish this is to add a penalty term to the cost function which penalises large weights. The idea is that large weights should only be used when absolutely necessary.

In theory, using this kind of regularisation should reduce the problems of overtraining with a too powerful network, e.g. a network with unnecessarily many hidden units. Check if this is true in this case. Use the performance function `msereg` and analyze the generalisation performance as a function of the number of hidden units again.

3.4 Generalization performance

We will now switch to a somewhat larger dataset, known as the *Protein Localization Sites database*, though it has been modified here by removing the mitochondrial location (MIT). The database contains attributes giving different scores obtained in various experimental tests and bioinformatics assays, along

with information about the coded localization of the protein. We will use this information to build a classifier that, given protein attributes, will tell us what it believes about the protein's localization in the cell. The data set is available in the file `yeast.data`, which can be loaded directly into `Matlab`. There are 1022 data points, one per row. The first eight columns in each row hold the input attributes, the last column contains the code for localization.

In this case, you should try the cases with one output node coding localization in its output value, as well one output node per localization (nine). Which setup works best? Show this in a graph where you have run a network, with some different number of hidden nodes, but one and nine output nodes respectively.

Select a reasonable network and network parameters and use the training method of your choice to train on 876 datapoints. Use the remaining points as test data to get an estimate of the generalization performance. Repeat this procedure, each time with different points as test data. Note that you should have no overlap between the different test sets, which means that you can only repeat this ten times. Calculate the average and standard deviation of the performance you get from these seven runs. What can these two values tell us? Try to disable Matlab's validation and see if you can increase performance, or if you see effects of overtraining.

Good luck!