nnn - Nick's Neural Network

Nicholas d'Alterio & Gideon Denby

July 11, 1995

Contents

1 Introduction

This program is an implementation of a fully connected feed forwards neural network with error back-propagation.

It offers the chance to train the network, which can be any size, in multiple stages with each stage containing up to 1000 training vectors. Once trained the network may be used to classify other vectors. The program is fully configurable and but simple to use.

2 Quickstarter

The configuration file included with the distribution is set up to train a neural network with the XOR problem. To use simply type:-

```
%> nnn < nnn.conf > nnn.trained.conf
```

This will train the neural network with the XOR vectors until 100% of the neural networks outputs agree with the target values to within a difference of 0.1.

The output file will contain an updated configuration file which will now have the final weights of the neural network stored. This can now be used to find out what the output of any of the XOR vectors should be. This is done by editing the configuration file and deleting the number at the end of the line in the last four lines of the file, then run the program as follows:-

```
%> nnn -i < nnn.trained.conf > nnn.classified.conf
```

This will print to the screen what the output of each of the vectors should be. In addition these values will also be placed in the new configuration file where the numbers were previously deleted.
For a list of command line options use the `-h` option.

3 How to use nnn

3.1 The Configuration File

This section describes the format of the configuration file for the nnn program.

The top few lines are generally comments which describe the file purpose and the purpose of the number below. The comment symbol is the # which appears at the start of a line indicating to the program that the rest of the line may be ignored.

The first non comment line contains the following numbers separated by spaces:-

1. The number of layer that the neural network will have. This number is generally set to 3 since a three layer network with more nodes is generally equivalent to network with more layers. This number is not restricted in its maximum size but should not be less than 2. It must be an integer.

2. The learning rate for the network. This is a floating point number which affects how the network learns. The higher the number the faster the network will be trained, however this has the side effect that the initial behavior is erratic and the final accuracy is less. If it is too high the network may never learn. Lower numbers mean slower training but more accurate results. The standard value used during test was 2.0

3. The gain factor. This is a floating point number which affects the slope of the sigmoid function, the higher the number the greater the slope. Increasing this may improve the network performance however taking it too high will cause the sigmoid to quickly saturate. Set this equal to 1.0 for standard neural network.

4. The momentum factor this is a floating point number in the range 0 to 1. Its purpose is to keep the network moving towards its final answer. It has the potential to vastly increase the rate of convergence but may also introduce extra local minima. Set the momentum factor to zero to go back to a tradition back propagation neural network.

5. A seed for the random number generator.

6. The accuracy that the neural network must achieve for the program to decide that it has learnt a particular vector correctly. This is a floating point number in the range 0 to 1, the smaller the number the more accurate.

7. The percentage of vectors that the network must learn before stopping the training process.

All the above options except for the number of layers may be overridden by command line options.

Next in the file is a marker `#Layers` which indicates that information about the network size follows. There must be as many lines as specified in the previous section number of layers.

Each line should contain an integer which is the number of nodes that particular layer in the network will contain with the top line being the input of the neural network. It is important that the number of nodes on the input layer and the number of nodes on the output layer match the number of inputs and outputs in the training vectors that will be applied to the network. There is no limitation on the size of the number here.
e.g.

```
#Layers                 Marker
2                       Number of nodes in input layer
2                       Number of nodes in hidden layer
1                       Number of output nodes
```

If the network has not been previously trained the next marker will be `#Vectors` otherwise it will be `#Weights` which will be followed by the a single line containing all the weights obtained during training. This is generated by the program.

The the end of the file in therefore always marked by the `#Vectors` marker which is followed by a number of lines each containing one vector. A vector consists of a bias, the input vector, and the expected outputs for that input vector. There is a limit of 1000 vectors in any run of the program.

The biases are extra normalising nodes for the neural network to prevent the output from getting out of control. Each layer of the network has one bias node and its value is given by the bias of the first vector, in addition the other vectors may have independent biases. In normal operation all biases are set to -1.
e.g.

```
#Vectors
-1               0    0              0
-1               0    1              1
-1               1    0              1
-1               1    1              0
bias         input vectors   expected output
```

NOTE The program currently does not have much error checking when parsing the configuration file. Therefore the user must ensure that the file is exactly as described above or there may be unpredictable results.

3.2 Training the Neural Network

To train the neural network first choose the network parameters and construct a configuration file with the training vectors at the bottom. Then run the program with the command:-

```
%> nnn < nnn.conf > new.nnn.conf
```

The program will then begin to train the network to the desired level of accuracy. If the network succeeds in the training it will print a message to the screen describing the percentage of vectors classified correctly and the number of iterations that it took to achieve it. If the network fails then a message will be printed stating the final classification level.

In both cases the weights will be saved in the file new.nn.conf which can then be used for classifying unknown vectors or further training.

An alternative to printing a configuration file is to print a graph of accuracy against number of iterations. This is done with the -g command line option.

NOTE The network will only classify output nodes to be either 0 or 1 so to train a neural network with a data set which has 2^n possible outcomes there must be n output nodes.

```
%> nnn -g < nnn.conf > graph.file
```

3.3 Classifying Unknown Vectors

For this it is necessary to have a configuration file containing the weights at the end of training. Edit this file and replace the vectors section with the set of biases and inputs that need to be classified. Then run the program as usual with the -m command line option.

```
%> nnn -i < nnn.trained.conf > nnn.classified.conf
```

This will classify each output node to be either 0 or 1 depending on the values after the vector to be classified has been fed through the network. These values will be printed both to the screen and to the file so that they follow the input part of the vector.

If no weights are present in the configuration file the program will print a message and exit.

3.4 Other Command Line Options

In addition to the command line options already mentioned there are a few command line options which enable the user to change the network settings without editing the configuration file. These are of the form

```
%> nnn -s 1024 < nnn.conf > nnn.new.conf
```

The above example sets the seed to be 1024. These command line options override any values found within the configuration file. They include:-

-r sets the learning rate

-m sets the momentum factor

-s sets the seed for the random number generator

-a sets desired accuracy for network to attain

-c sets the percentage of vectors to classify correctly

-k sets the gain factor

3.5 Problems

There may be occasions where the neural network will not converge towards the expected output and there are often many reasons for this. This section will attempt to make some suggestions to improve the chances of correct training.

1. Use the -g option to see how the training proceeds.

2. Try changing the learning rate. A low learning rate may become stuck in a local minima whilst a high rate may never settle down.

3. Adjust the momentum factor. This may increase the speed of convergence of the network.

4. It may be that the random series generated by the seed gets the network stuck.

5. Think about reducing the accuracy that is expected from the network, there will be situations where it is just not possible to get very accurate results.

6. Reduce the percentage of vectors that is expected to be classified correctly during training. It is often the case that all but a couple of vectors will classify.

7. When using binary vectors try having 0 as 0.1 and 1 as 0.9 this moves the output away from the flat area on the sigmoid function and increase the chance of convergence.

8. It may be that there are not enough hidden nodes to correctly define the problem. Increasing the number of hidden nodes generally speeds up convergence to the correct output.

9. Do NOT expect to train to get an output of greater than one. This is outside the range of the sigmoid and will therefore NEVER be successful.

10. If the sigmoid is saturating make this value small so that the slope is decreased.

11. As a last resort it may be worth adjusting the value of the biases however this is not recommended.

There are cases where this type of neural network is not appropriate and there are certainly cases where this is not the best solution so experiment a bit.

4 Bugs

At present there are no known bugs. However there are circumstances in which the program will crash if not used correctly.

For example if an option which requires a numeric argument is used without the argument a non-existent array element will be accessed.

If the program is used as detailed in this document there will be no problems.

If you find a bug or edit this program please send details to n.dalterio@ic.ac.uk

5 History

This is out third attempt at writing a neural network program and was designed to be easy to read and understand unlike all other neural network programs we have managed to obtain in source form.

The main reason for the problems in getting a working neural network program originate with poorly written books on the subject which don't make there notation clear and consequently have impossible to understand algorithms.

In addition there are many points which are not mentioned in books but are essential to getting the program to work correctly.

6 Release Conditions

There are none, but I accept no responsibility for things that may happen because to its use.

7 A Quick Plug

This program was created as part of a project for my Physics degree , for the full report and other work I have done see

`http://crab.sp.ph.ic.ac.uk/~projects/nick_gid/index.html`