Neural Networks for Multiple Class Object Detection Package:

USER GUIDE

Mengjie Zhang

Department of Computer Science
Royal Melbourne Institute of Technology

14 Feb 2000

2.1.3 multipat

```
% multipat  pattern class1 class2 ... classN
```

The *multipat* program creates network pattern files from PGM images stored in multiple directories. The first argument to multipat gives the name of the pattern file to create. Up to eight additional arguments follow, which represent examples of each class to be created.

The class files are actually text files that contain the locations of PGM images belonging to each class.

For example, if we have four different classes of image that we'd like the network to distinguish between, we must create four directories, one for each class, and populate each directory with example images from that class. For each directory, a text file is then created which contains a listing of all of the files in the directory. This listing is given as an argument to multipat, which in turn reads the listing and loads the appropriate PGM images. The multipat program alternately reads an image from each class, copying the data to the pattern file, thus ensuring that the data can be trained in sequential or batch mode learning.

2.2 Network Training Tools

The *nntrain* program trains a neural network. The *nntest* program is used to verify the network's performance on data not used in the training phase.

2.2.1 nntrain

```
% nntrain network pattern [weights]
```

The *nntrain* program teaches a neural network to associate input patterns with output patterns. The network and pattern arguments to nntrain are mandatory. The formats of these files will be described in section 3. The third argument to *nntrain* is an optional weights file, which may have been previously produced by *nntrain*. This enables stopped training runs to be continued.

There are four conditions under which network training will be stopped:

1. The *user control* strategy. The user presses CTRL-C, which stops training and causes the weights to be saved to a file *weights.dat*.

2. The *error control* strategy. The network produces an error less than critical_error, which indicates learning to the specified error has completed.

3. The *epoch/cycle control* strategy, where the training will keep going until the training epochs/cycles reach a given number.

4. The *proportion control* strategy. When the proportion of the number of patterns "correctly "classified among the number of total training set reach a pre-defined percentage, the train will be stopped. The *classification criterion* is that the neural network classification is regarded as "correct" if the class whose actual neural (output) activation value is the biggest one among all the classes is the same as the desired class whose desired neural output is 1.0. Otherwise, this pattern is "incorrectly" classified.

The *nntrain* software displays the current network error and the classification accuracy on stdout.

Please note that there is another similar program called *nntrain-epoch*, where the weights change after every epoch rather than every pattern in the program *nntrain* presented above.

2.2.2 nntest

```
% nntest network pattern weights
```

The *nntest* software provides a report (written to stdout) on the network's performance on both the training and test data contained in the supplied pattern file. All arguments are mandatory.

2.3 Object Detection Tools

For each class, the *nnsweep-img-loc* program is used to apply the trained neural network generated by *nntrain* as the template, in a moving window fashion, over a large image in order to locate the objects of interest. The template is swept across and down the large picture pixel by pixel in every possible position. The *shr-find-center* program is used for finding the centers of the detected objects. We use the centers of the objects to represent the corresponding objects themselves.

2.3.1 nnsweep-img-loc

```
% nnsweep-img-loc network weights image
```

The *nnsweep-img-loc* software is used for network sweeping including object localisation and classification experiments. *nnsweep-img-loc* takes a network description file, a weights file, and an image file as arguments. Typically the neural network being passed to *nnsweep-img-loc* has been taught (trained) to perform object classification task (with the input patterns of the cut-outs). This network is then applied to the image file, a region with the size of the input field at a time in every possible pixel position in the image.

A PGM format image and a position file are produced for each class of interest. A PGM image for the *other* class (background) is also generated.

3.4 network definition files (.net)

The network definition file specifies all aspects of a networks architecture and training parameters. Note that the critical error here is the *mean squared error*, rather than *total sum squared error*.

```
lr learning_rate
m momentum_value
ce critical_error
r random_range
percent classification_accuracy
number_of_layers
units_in_input_layer
units_in_hidden_layer
units_in_output_layer
```

For example, the network file *images.net* for easy pictures is as follows:

```
lr 0.5
m 0.0
ce 0.002
r 0.5
percent 100.0
3
196
3
4
```

The network definition template given above shows a 3-layer network architecture. More layers are possible - just more hidden layer definitions.

3.5 pattern files (.pat)

Pattern files are produced by the *multipat* series of commands. See Pattern Generation Tools for more information. Note that the pattern files used here are usually binary. We also have the program of *multipat-ascii* which can produce a pattern file with the full text format. Each pattern contains an input pattern and an output pattern. The pattern file format is given as follows:

```
0.549  0.549 ... 0.867  0.690  0.988  0.753 ... 1.00 0.00 0.00 0.00
0.788  0.549  0.953  0.788 ... 0.914  0.549 ... 0.00 1.00 0.00 0.00
0.055  0.051  0.047 ... 0.075  0.035  0.071 ... 0.00 0.00 1.00 0.00
0.549  0.549  0.549 ... 0.549  0.549  0.549 ... 0.00 0.00 0.00 1.00
...    ...
```

3.6 weight files (.dat)

Weight files consist of floating point numbers represented in ASCII. The weight are given in linear order; the weight from input node 0 to hidden node 0 is given first, followed by the weight from input node 1 to hidden node 0, etc... The last weight given is from the last hidden node to the last output node. Please note that the nodes in the input layer do not have biases, or the biases are 0.

Following the weight values are the bias values. Each computational unit that produces an output (ie: hidden and output units) will have a bias value associated with it listed at the end of the weights file.

A part of the *weights.dat* for the easy pictures is given below.

```
0.014683
0.012763
0.014847
 ... ...
-0.212967
-0.210692
-0.196092
 ... ...
0.000000
0.000000
0.000000
```

Weight files are produced by the *nntrain* programs.

3.7 .nam file

This type of files gives the names of all the object classes in a database. They are used for network sweeping and called by the *nnsweep-img-loc* program in the command line. An example for the detection of easy pictures *objects.nam* is given as follows:

```
class1
class2
class3
other
```

3.8 location file and center files (.txt) and (.doc)

3.8.1 Location file format

A part of the location file for class1 *class1_loc.txt* in easy pictures is as follows: