Homework 5B: Face the Network
Additional Documentation

15-681: Machine Learning
Roni Rosenfeld
Carnegie Mellon University

October 21, 1999

1 Additional documentation

There are several utilities provided with this assignment:

- `nettrain`: The neural network training utility. It utilizes the back-propagation algorithm. You will make some small modifications to this code in the file "imagenet.c".
- `netsense`: The network sensitivity analysis visualization utility. It is not necessary to modify anything here.
- `imontage`: The image "montaging" utility. It is not necessary to modify anything here.
- `nettopgm`: The hidden unit weight visualization utility. It's not necessary to modify anything here.

The only code which needs to be modified in this assignment is in the file "imagenet.c". Do not modify any other code unless you are working on the optional section and in that case make changes to a separate copy of the code. For the required section of the assignment, your "imagenet.c" file ***must*** work with the original version of the other files or no credit can be given.

If you have any questions about this assignment please contact Chuck Rosenberg, `chuck@cs.cmu.edu`.

1.1 `nettrain`

1.1.1 Background

The `nettrain` program implements a neural network learning package. The learning algorithm used is the stochastic variant of backpropagation. The program can accept data in the form of lists of images, in which case the target function and number of output units is determined by the functions in the file "imagenet.c" and the target configuration option or as text based data files.

1.1.2 Running `nettrain`

The `nettrain` program has several options which can be specified on the command line. This section briefly describes how each option works. A very short summary of this information can be obtained by running `nettrain` with no arguments. The list of options is as follows:

`-o <output network file>` - The trained neural network will be saved to this file.

`-i <input network file>` - This option loads an existing network file.

`-e <number of epochs>` - This option specifies the number of training epochs which will be run. If this option is not specified, the default is 100.

`-n <eta, the learning rate>` - This option specifies "η", the learning rate, in the weight update rule. If this option is not specified, the default is `0.2`.

`-m <the momentum term>` - This option specifies the momentum term scaling factor in the weight update rule. If this option is not specified, the default is `0`.

`-u <the number of hidden units>` - This option specifies the number of hidden units in the network. A `0` value signifies a network with no hidden units, the input units are connected directly to the output units. If this option is not specified, the default is `4`.

`-s <seed>` - This option specifies an integer which will be used as the seed for the random number generator. The default seed is `101299`. This allows you to reproduce experiments, if necessary, by generating the same sequence of random numbers. It also allows you to try a different set of random numbers by changing the seed.

`-S <number of epochs between saves>` - This option specifies the number of epochs between saves. The default is `0`, which means that no intermediate networks will be saved.

`-R <number of epochs between network reports>` - This option specifies the number of epochs between network performance evaluations. The default is `10`. Be careful when using this option, the error on the validation set is only computed when a performance evaluation occurs. Which means we will only capture the best network to the nearest of 10 epochs, which is OK for the purposes of this assignment.

`-t <training set file>` - This option specifies the name of the *training set* file. This can either be a text file which contains a list of image pathnames or a numeric data set. The program will automatically determines which type of file it is by examining its contents. This option must be specified.

`-1 <test set 1 file>` - This option specifies the name of the *validation set* file. If this option is not specified, no statistics for the validation set will be generated.

`-2 <test set 2 file>` - Same as above, but for the *test set*.

`-c <configure image target value>` - This option specifies an integer which will be used to select a computed target value when image based data sets are used. The default is `1`.

`-E <final summary of errors only>` - This option causes the code to output the network output values for just the data set examples which were misclassified.

`-F <full final summary>` - This option causes the code to output the network output values for all data set examples.

`-N <turn off image normalization>` - This option disables the normalization of images during loading. By default, images are normalized when loaded to eliminate variations in overall brightness. This is done via a linear transformation which ensures that the mean and variance of the central region of each image will be the same across images.

`-M <image mask config>` - This option selects an image masking option. The current option values are: `0` – no masking, `1` – mask the top part of the image, `2` – mask off all but the face section of the image. The default is `0`.

Here is an example call:

```
./nettrain -n 0.2 -u 0 -c 1 -e 1500 -o shades.net -t straightrnd_train.list
          -1 straightrnd_test1.list -2 straightrnd_test2.list
```

1.1.3 Interpreting the output of `nettrain`

When you run `nettrain`, it will first read in all the data files and print a bunch of lines regarding these operations. Once all the data is loaded, it will begin training. At this point, the network's training and test set performance is outlined in one line per epoch. For each epoch, the following performance measures are output:

```
<Epoch> <Wt Mag> <Trn Acc> <Trn Mse> <Val Acc> <Trn Mse> <Tst Acc> <Tst Mse>
```

These values have the following meanings:

`Epoch` This is the number of the epoch just completed. A value of 0 means that no training has yet been performed.

`Wt Mag` This is the average of the absolute values of all the weights in the network.

`Trn Acc` This is the percentage of examples in the training set which were correctly classified. An example is considered correctly classified if every network output is within 0.4 of the target value, that is $\forall_i \, |t_i - o_i| \leq 0.4$, where t_i is the target value for output unit i and o_i is the actual output value for that unit.

`Trn Mse` This is the average of the mean squared error over all training examples. The mean squared error is calculated as $\frac{\sum (t_i - o_i)^2}{n}$, where t_i is the target value for output unit i and o_i is the actual output value for that unit, and n is the number of output units.

`Val Acc` This is the percentage of examples in the validation set which were correctly classified.

`Val Mse` This is the average, over all examples in the validation set, of the mean squared output error.

`Tst Acc` This is the percentage of examples in the test set which were correctly classified.

`Tst Mse` This is the average, over all examples in the test set, of the mean squared output error.

1.1.4 The Code

The code for `nettrain` is broken into several modules:

- `nettrain.c`: The top-level program which uses all of the modules to implement neural network training. **You will not need to modify any code in this module to complete the assignment.**

- `imagenet.c`: The routines for setting up target vectors for training and selecting the "best" network to save. You will need to modify code in this file .

- `pgmimage.c`, `pgmimage.h`: The image package. Supports read/write of PGM image files and pixel access/assignment. Provides an `img_t` data structure for images, and an `imgl_t` data structure for image lists (an array of pointers to images; useful when handling many images). **You will not need to modify any code in this module to complete the assignment.**

- `backprop.c`, `backprop.h`: The neural network package. Supports three-layer fully-connected feedforward networks, using the backpropagation algorithm for weight tuning. Provides high level routines for creating, training, and using networks. **You will not need to modify any code in this module to complete the assignment.**

- `util.c`: A collection of general purpose routines.

1.2 `netsense`

The `netsense` program performs a sensitivity analysis of a trained network utilizing a specific data file. This is done by presenting each of the examples to the network, recording the network output, perturbing the network's inputs one by one and analyzing the changes in the outputs those perturbations cause. (The size of the perturbation for each input is the standard deviation of that input as computed from the data set provided.) This perturbation analysis is performed for each of the examples in the data set and the results are averaged together. The final result is output as an PGM image where lighter pixels represent inputs that have a larger effect on the output and darker pixels inputs that have a lesser effect. When a sensitivity analysis is performed on the hidden units, the resulting sensitivity values are scaled by the average of the absolute values of the weights from those hidden units to the output units. When multiple units are being analyzed, a single image is output in which each sub-image is the result of a single sensitivity analysis.

The `netsense` program has several options which can be specified on the command line. This section briefly describes how each option works. A very short summary of this information can be obtained by running `netsense` with no arguments. The list of options is as follows:

`-i <input network file>` - This option specifies the name of an existing network file to load. This option must be specified.

`-t <test set file>` - This option specifies the name of the test set file. This can either be a text file which contains a list of image pathnames or a numeric data set. The program will automatically determine which type of file it is by examining its contents. This option must be specified.

`-O <output image file for output units>` - The sensitivity analysis for the output units will be saved to this file.

`-H <output image file for hidden units>` - The sensitivity analysis for the hidden units will be saved to this file.

`-w <width of unit weight plot>` - The width of each unit sub-image, automatically determined for image test data.

`-m <integer output image magnification>` - This option specifies an integer scaling factor for the output images. This is helpful when the input images are quite small.

`-N <turn off image normalization>` - This option disables the normalization of images during loading. By default, images are normalized when loaded to eliminate variations in overall brightness. This is done via a linear transformation which ensures that the mean and variance of the central region of each image will be the same across images.

`-M <image mask config>` - This option selects an image masking option. The current option values are: `0` – no masking, `1` – mask the top part of the image, `2` – mask off all but the face section of the image. The default is `0`.

Here is an example call:

```
./netsense -i shades1.net -O shades1.pgm -t straightrnd_test2.list
```

1.3 `imontage`

The `imontage` program takes a list of PGM images as input and generates a new PGM image which is the combination of all of the images into a single image, a "montage". This is convenient for visualizing the content of a large group of images.

The `imontage` program has several options which can be specified on the command line. This section briefly describes how each option works. A very short summary of this information can be obtained by running `imontage` with no arguments. The list of options is as follows:

`-i <input list of images>` - A file with a list of images to montage. This option must be specified.

`-o <output image file>` - The image montage will be saved to this file. This option must be specified.

`-s <inter-image spacing>` - The number of pixels between sub-images. The default value is `3`.

`-m <integer output image magnification>` - This option specifies an integer scaling factor for the output images. This is helpful when the input images are quite small.

`-M <image mask config>` - This option selects an image masking option. The current option values are: `0` – no masking, `1` – mask the top part of the image, `2` – mask off all but the face section of the image. The default is `0`.

`-N <turn off image normalization>` - This option disables the normalization of images during loading. By default, images are normalized when loaded to eliminate variations in overall brightness. This is done via a linear transformation which ensures that the mean and variance of the central region of each image will be the same across images.

Here is an example call:

```
./imontage -i straightrnd_train.list -o train1.pgm
```

1.4 nettopgm

The `nettopgm` program dumps either the weights from the inputs to the hidden units or the weights from the hidden units to the outputs in the specified network to a PGM image file. (When a network has no hidden units, the weights from the input units to the output units are dumped.) When there are multiple hidden or output units, an image is dumped which consists of a montage of sub-images, one for each unit. The values are scaled before being plotted as images such that the maximum value will fill the range of 0-255. The absolute values of the weights or their original signed values can be dumped. When absolute values are dumped, a weight with a value of "0" will be mapped to black (a pixel value of 0) and the weight with the largest absolute value which will be mapped to white (a pixel value of 255). When signed values are dumped, a weight with a value of "0" will be mapped to mid-level gray (a pixel value of 128), negative values are mapped to values less than 128 (darker) and positive values are mapped to values more than 128 (lighter).

The `nettopgm` program has several options which can be specified on the command line. This section briefly describes how each option works. A very short summary of this information can be obtained by running `nettopgm` with no arguments. The list of options is as follows:

`-i <input network file>` - This option specifies the name of an existing network file to load. This option must be specified.

`-o <output image file>` - The unit weight plot image will be saved to this file.

`-w <width of unit weight plot>` - The width of each unit sub-image. The default is 32.

`-H <dump hidden unit weights>` - Selects the hidden unit weights as the ones to dump.

`-O <dump output unit weights>` - Selects the output unit weights as the ones to dump.

`-A <dump absolute values>` - The weight dump will represent the absolute values of the weights in the network.

`-S <dump signed>` - The weight dump will represent the actual signed values of the weights in the network.

`-m <integer output image magnification>` - This option specifies an integer scaling factor for the output images. This is helpful when the output images are quite small.

Here is an example call:

```
./nettopgm -i shades1.net -o shades1-wts.pgm -A -O
```