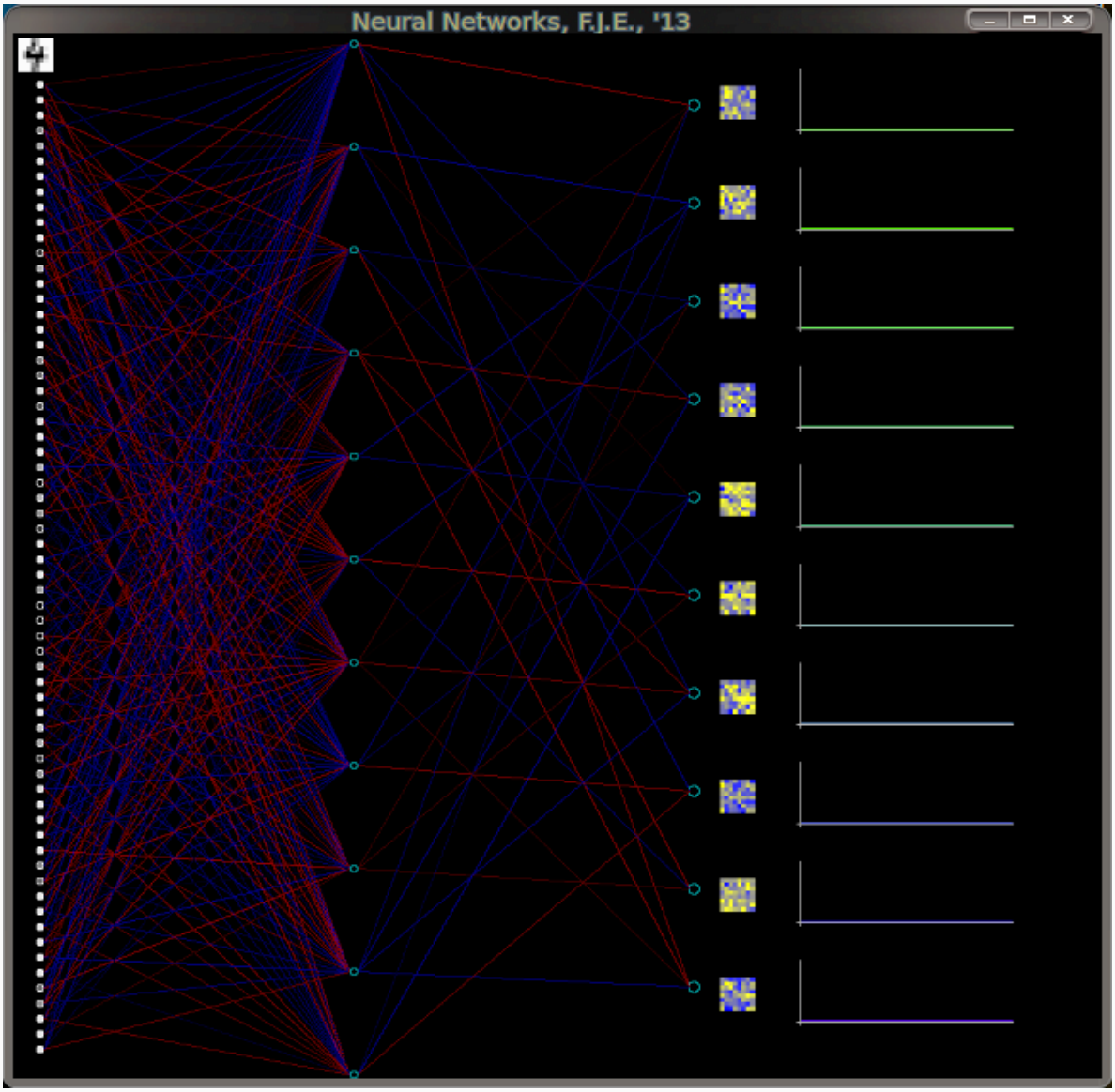
Assignment 5
One Net to Rule Them All

Due date: Monday, Mar. 25th at 9am
(electronic submission on Mathlab)

This assignment is worth 15 units toward the total assigned work you will do during the term



Neural Networks – Optical Character Recognition

The goal of this assignment is to let you implement the main components of a neural-network training process. You will experiment with network architecture, try out different activation functions, and become familiar with the information flow over the neural network. You will have to understand how the weights in the network are trained, and once you have a complete implementation, you will be able to observe the network learn in real-time.

Learning Objectives:

You will understand the structure of one and two-layer neural networks. You will observe the layout and connections between neurons, and see the network in action.

You will strengthen your understanding of the information flow within the network. Both in the forward direction (outputs), and in the backward direction (errors).

You will understand how error back-propagation works. You will carry out the back-propagation process in order to adjust the weights linking neurons together.

You will experiment with different network layouts and activation functions, and through this process gain experience about how different neural networks behave.

You will explore a simple (but realistic!) optical character recognition problem that nevertheless shows how more general systems could be built to solve harder problems.

Skills Developed:

Managing neural network structure. Understanding how weights are encoded, stored, and used to compute outputs.

Computing error signals from training data, and using them to adjust the network's weights.

Testing multiple layouts to find one that works best. Comparing the performance of different network architectures.

Reference material:

Your in-class lecture notes on Neural Nets

Digit recognition dataset:

Optical Recognition of Handwritten Digits

E. Alpaydin, C. Kaynak
Department of Computer Engineering
Bogazici University, 80815 Istanbul Turkey
alpaydin@boun.edu.tr
July 1998

Distributed by the UCI Machine Learning repository: http://archive.ics.uci.edu/ml/

Optical Character Recognition

Automatic conversion from hand-written text to electronic format is an important application of AI and computer vision. Here you will study a very carefully constrained version of this problem: You will train a Neural Network to recognize hand-written digits such as may appear on mailing envelopes for postal codes, street addresses, and so on.

The specific dataset we will use has been widely studied. It consists of digitized images of digits that have been pre-processed so that the digits all have the same size (8x8 pixels) and a uniform range of brightness values. There is a couple thousand digits, along with associated labels which indicate what digit a particular image represents.

As an OCR problem, this is a reasonably easy one. But the principles you will study in this assignment apply to more general and more difficult OCR problems, as well as to general classification tasks. In particular, Google is currently using Neural Nets for text-to-speech processing on Android devices! (http://research.google.com/pubs/SpeechProcessing.html)

While working on this assignment, think about how you would solve a more general OCR problem in which characters have not already been cut and shrunk to a uniform size, and how you would handle variations in handwriting styles (which are much more pronounced for letters than numbers).

Step 1

Download and uncompress the starter code into an empty directory.

This code is designed for Linux. I recommend you do all your work on the CS lab at IC 406. You can work remotely by logging into ***mathlab.utsc.utoronto.ca*** but note that the program uses OpenGL graphical display and will run very slowly over ssh.

We will not be able to support Windows, if you choose to develop on Windows you will have to set up your computer properly, and you ***must*** ensure your final submission ***works on mathlab***.

The starter code contains the following files:

NeuralNets_core_GL.pyc
NeuralNets_global_data.pyc
NeuralNets_train.py
REPORT.TXT

The only program file you need to look at is ***NeuralNets_train.py***. All your code will be written here.

Do not add any extra Python program files. Also, you ***can not add any global data***.

Read all the comments in NeuralNets_train.py. The comments describe the global data that is available to you and tell you what needs to be implemented.

Feel free to ask for clarification at any time. But I expect you to have ***read this handout and all the comments in the code carefully***.

Neural Nets for OCR

Step 2

Test-run the starter code:

1) Start an interactive Python shell

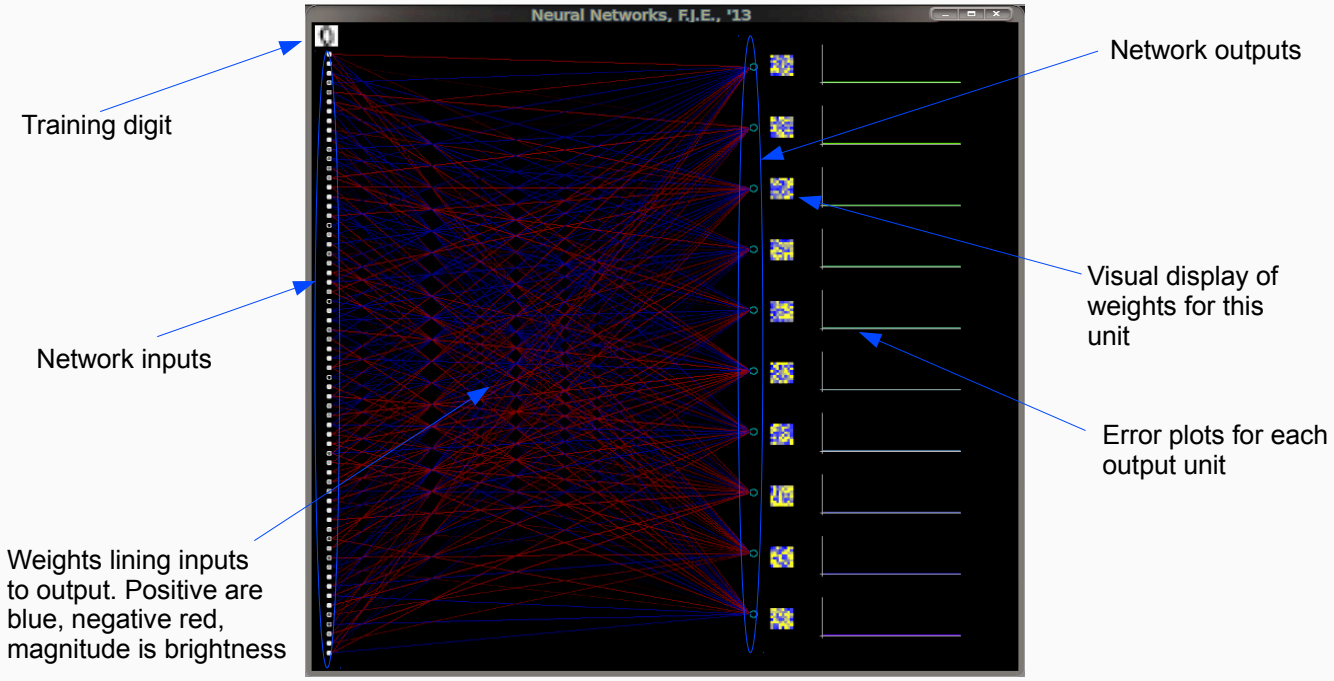
2) On the shell prompt execute:

```
>>>from NeuralNets_core_GL import *
>>>NeuralNets_init(0,0,1.0,1234)
>>>NeuralNets_train(.05)
```

The first line initializes the network and defines its atchitecture. There are 4 input parameters:
- Number of hidden-layer units. Zero here means a single-layer network
- Type of sigmoid activation function. 0 → logistic, 1 → hyperbolic tangent
- Learning rate in (0, 10.0]
- Initial random seed

The second line starts the training/testing process. The input parameters specifies the threshold on training error that will be used to stop training, i.e. this value determines when the network is 'good enough'. Once the error has gone below this threshold, the code computes and reports classification accuracy.

You will work first on networks with ***no hidden layer***. Therefore, you should start with zero hidden units. Launching the above will produce an image like the one below:



Neural Nets for OCR

Step 3

The code initially doesn't do much. It runs through a set of digits, and outputs classification rates given the initial weights. But of course, neither the output computations nor the weight adjustments are taking place.

Implement your solution. You will write the following components:

- Sigmoid activation functions
- Derivatives of sigmoid functions
- Feed-Forward computation for outputs
- Error computation and error back-propagation
- Weight adjustments

The starter describes what needs to be implemented and where. ***Note there will be two slightly different feed-forward and error adjustment blocks.*** One for networks with no hidden layers, and a different one for two-layer nets. The weights are stored in different arrays for these two architectures. Make sure you use the correct weight array.

Answer the relevant questions and carry out the tests described in the ***REPORT.TXT*** file. note that you will ***need to capture a couple screenshots and submit them with your code.***

What to expect:

If you implemented things properly you should see:

- Weights changing during training. Initially the changes will be quite visible, as the network becomes better at the task the changes will be more subtle.

- The error plots should show a decreasing trend (though they will be noisy!)

- The average squared error (printed on the terminal) should have a decreasing trend (also noisy, it may oscillate a bit)

What you will need to play with:

The learning rate and the type of sigmoid function

The threshold used to terminate training. Lower error thresholds involve longer training (and it is not guaranteed a given threshold will be reached), but they also lead to higher classification accuracy.

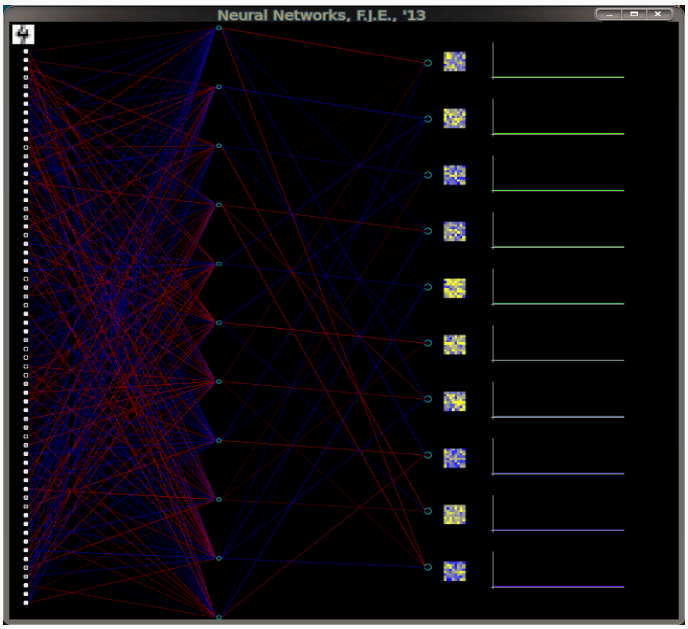
Pay attention to the average squared error reported during training. Use that to select a reasonable stopping threshold.

Neural Nets for OCR

Step 4

Once you have a working solution for the single-layer network, implement the corresponding feed-forward computation and back-propagation for the two-layer networks. This is only a bit more involved than what you have already done, but be careful to get the error back-propagation right!

The network structure looks like the image below. You will now see the '***hidden***' layer (not hidden from us, is it?), and the weights connecting units together. ***Note:*** Not all the weights are drawn (there is too many). The code only draws a subset of all weights in the net. This is also true for the one-layer nets.



Answer all the questions and carry out all the tests described in the ***REPORT.TXT***

What to expect:

- Similar to one-layer nets: Decreasing trend in error plots and reported square error

- Visible weights updates at the beginning, followed by subtle adjustments

What you will need to play with:

The learning rate and the type of sigmoid function. This is more critical for two-layer networks, so try out a few values, and test smartly.

The threshold used to terminate training. Again this is a bit more tricky for two-layer nets, but you should be able to figure out how far you can push it by looking at the reported square errors over time.

Neural Nets for OCR

Step 5

Submit your work:

Create a single compressed ***.zip*** file – and that means it should actually be ***in .zip format***. If you submit a compressed tar file, a .ar, .arj, .arc, .7z, or anything else renamed to have the .zip extension you will incur the wrath of your TA and lose marks.

The .zip file should contain:

```
NeuralNets_train.py
REPORT.TXT
NeuralNets_core_GL.pyc
NeuralNets_global_data.pyc
1layer-logistic-final.jpg
1layer-tanh-final.jpg
2layer-logistic-final.jpg
2layer-tanh-final.jpg
```

Your .zip file should be named:

```
NeuralNets_studentNo.zip
```

(e.g. NeuralNets_012345678.zip)

Submit your file on mathlab with the command:

```
submit -c cscd84w13  -a A5 -f NeuralNets_studentNo.zip
```

Before you submit, make sure your compressed file in fact contains all your files and code, and that it decompresses properly on mathlab. Non-working .zip files will get zero marks.